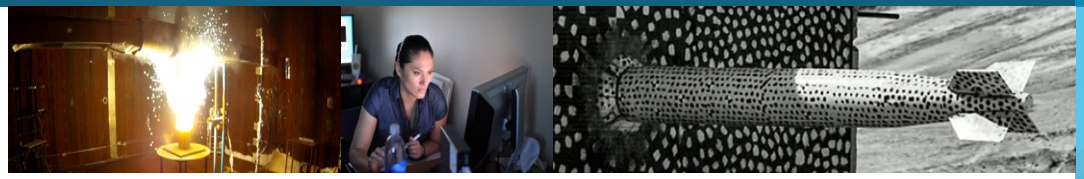
Asynchronous preconditioners and linear solvers



Erik G. Boman
Sandia National Labs

ICERM Seminar, Providence, RI
May 5, 2026

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Background

Asynchronous: A set of processors (e.g. CPU cores, GPUs, etc) work independently in parallel and without global synchronization

- Use most recent information, no idle time

Current interest: Communication & synchronization have become increasingly expensive

Much work on asynchronous linear solvers:

Chazan & Miranker [1969], Baudet [1978], Bertsekas [1983], Bertsekas & Tsitsiklis [1989], Frommer & Szyld [2000], Avron et al. [2015], …

"Asynchronous" means different things to different people

1. Asynchronous execution, but algorithm is mathematically the same
   - Generally accepted by users
2. Asynchronous execution, but algorithm is mathematically different!
   - Execution order matters
   - Results are non-deterministic, not reproducable
   - Problematic, many users don't like it!

Krylov Subspace methods:

Krylov methods (CG, MINRES, GMRES) are typically the preferred methods in synchronous parallel computing.

Challenge: They all require global inner products. This is bad for asynchronous methods!

Alternative: Stationary iterative methods

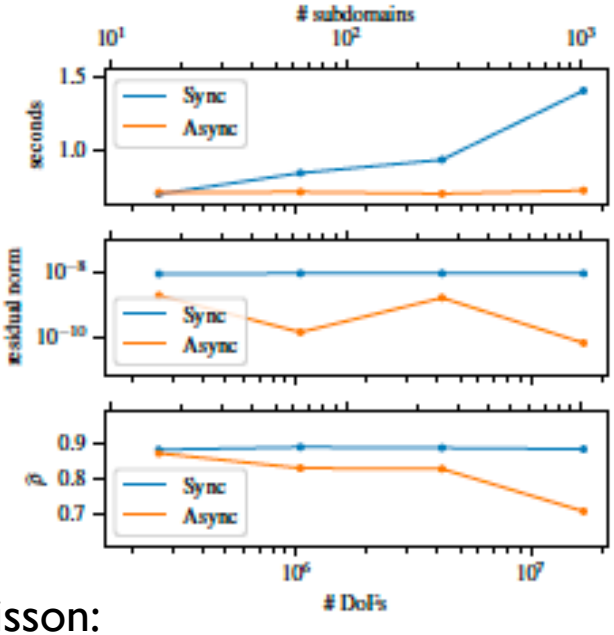
- (Preconditioned) Richardson
- Domain decomposition (Schwarz)
- Multigrid
- These can be implemented asynchronously, but typically converge slower than Krylov methods

Additive Schwarz (2-level)

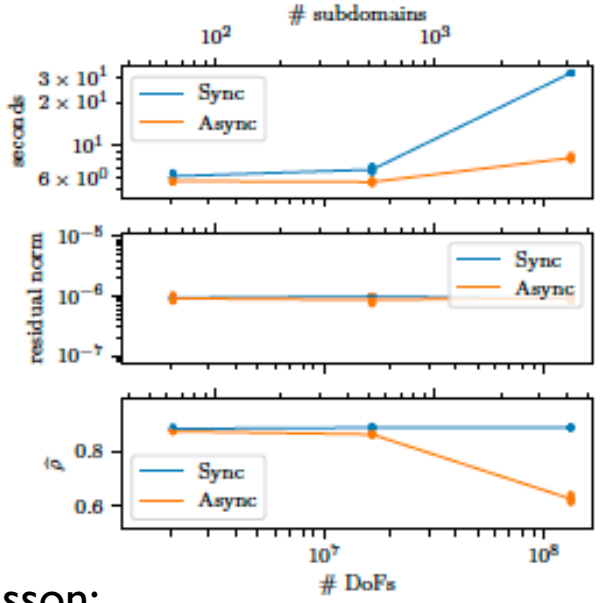
RAS (any Schwarz) is simple to implement as asynchronous solver, but not scalable

Two-level RAS is tricky to implement due to the coarse level correction, but scalable

Weak scaling from Glusa et al., SISC 2020.



2D Poisson:



3D Poisson:

Partially Asynchronous Solvers

Preconditioner setup is asynchronous

Preconditioner apply may be synchronous(?)

Use Krylov methods (synchronous)

Benefit: Incremental approach, less disruptive

Drawback: Much of the computation is synchronous

Asynchronous Preconditioners

1. Stationary iteration as preconditioner (Jacobi, Gauss-Seidel, Schwarz, etc.)
   - Preconditioner is not fixed, need flexible Krylov (FCG, FGMRES)
2. Factorization-based preconditioners (ILU)

   Preconditioner is fixed, can use CG/GMRES
   - Except if we use inexact (iterative) triangular solves

Incomplete factorizations (ILU)

The standard ILU methods compute similar to Gaussian elimination

- Highly sequential
- Limited opportunities for parallelism

Iterative ILU

- Compute L and U iteratively
- Highly parallel
- ParILU (Chow & Patel, 2015)
  - Fixed-point iteration, can be asynchronous

Iterative ILU

ILU(k) factorizations satisfy the property:

$$(A - LU)_{ij} = 0 \; \forall (i,j) \in S$$

ParILU solves these equations by asynchronous Jacobi or by Gauss-Seidel (synchronous version).

Note that these systems for L and U are actually larger than the original system with A!

Inexact solutions suffice; typically 3-4 sweeps is enough

A slight generalization is to minimize (pattern-constrained) ILU residual:

$$\min_{L,\,U} \| A - LU \| \text{ or } \min_{L,\,U} \| S \circledcirc (A - LU) \|$$

ATS-ILU: Overview

ATS: Alternating Triangular Solves

Key idea: $A \approx LU$, alternately update L and U

- Full (exact) triangular solves are too expensive

Instead:

- Update one row (L) or one column (U) at a time
- Extract relevant nonzero parts (small), do SpTRSV
- Highly parallel: All rows can be updated simultaneously

Theorem: ATS-ILU is a block-Newton method for minimizing the pattern-constrained ILU residual (Tunnell & B. 2026)

ATS-ILU: Algorithm

Algorithm 6.1 Synchronous ATS-ILU (block-Newton sweep on $R(\boldsymbol{L},\boldsymbol{U},\mathcal{S})=\mathbf{0}$)

Require: $\boldsymbol{A}$; sparsity patterns $\mathcal{L},\mathcal{U}$ with $\mathcal{S}=\mathcal{L}\cup\mathcal{U}$; feasible $(\boldsymbol{L},\boldsymbol{U})\in\mathbf{L}\times\mathbf{U}$

1: **for all** $i=1,\dots,n$ **in parallel do** ▷ *Row sweep (update L, keep U fixed)*

2: $\quad\mathcal{P}_i\leftarrow\mathrm{supp}(\mathcal{L}_{i,:})$; $\quad\boldsymbol{L}_{i,\mathcal{P}_i}\leftarrow\boldsymbol{a}_{i,\mathcal{P}_i}\boldsymbol{U}_{\mathcal{P}_i}^{-1}$

3: $\boldsymbol{D}\leftarrow\mathrm{diag}(\boldsymbol{L})^{-1}$; $\quad\boldsymbol{L}\leftarrow\boldsymbol{L}\boldsymbol{D}$ ▷ *Gauge fix (unit L, U is implicit)*

4: **for all** $j=1,\dots,n$ **in parallel do** ▷ *Column sweep (update U, keep L fixed)*

5: $\quad\mathcal{Q}_j\leftarrow\mathrm{supp}(\mathcal{U}_{:,j})$; $\quad\boldsymbol{U}_{\mathcal{Q}_j,j}\leftarrow\boldsymbol{L}_{\mathcal{Q}_j}^{-1}\boldsymbol{a}_{\mathcal{Q}_j,j}$

All block solves are restricted to principal submatrices aligned with $\mathcal{L},\mathcal{U}$; updates are therefore exact on the pattern and require no projection.

Algorithm 6.2 Asynchronous ATS-ILU (interleaved block-Newton updates)

Require: $\boldsymbol{A}$; sparsity patterns $\mathcal{L},\mathcal{U}$ with $\mathcal{S}=\mathcal{L}\cup\mathcal{U}$; feasible $(\boldsymbol{L},\boldsymbol{U})\in\mathbf{L}\times\mathbf{U}$

1: **for all** $t=1,\dots,n$ **in parallel do** ▷ *Fair selection; bounded staleness allowed*

2: $\quad\mathcal{P}_t\leftarrow\mathrm{supp}(\mathcal{L}_{t,:})$; $\quad\boldsymbol{L}_{t,\mathcal{P}_t}\leftarrow\boldsymbol{a}_{t,\mathcal{P}_t}\boldsymbol{U}_{\mathcal{P}_t}^{-1}$

3: $\quad\mathcal{Q}_t\leftarrow\mathrm{supp}(\mathcal{U}_{:,t})$; $\quad\boldsymbol{U}_{\mathcal{Q}_t,t}\leftarrow\boldsymbol{L}_{\mathcal{Q}_t}^{-1}\boldsymbol{a}_{\mathcal{Q}_t,t}$

ATS-ILU: One ATS step

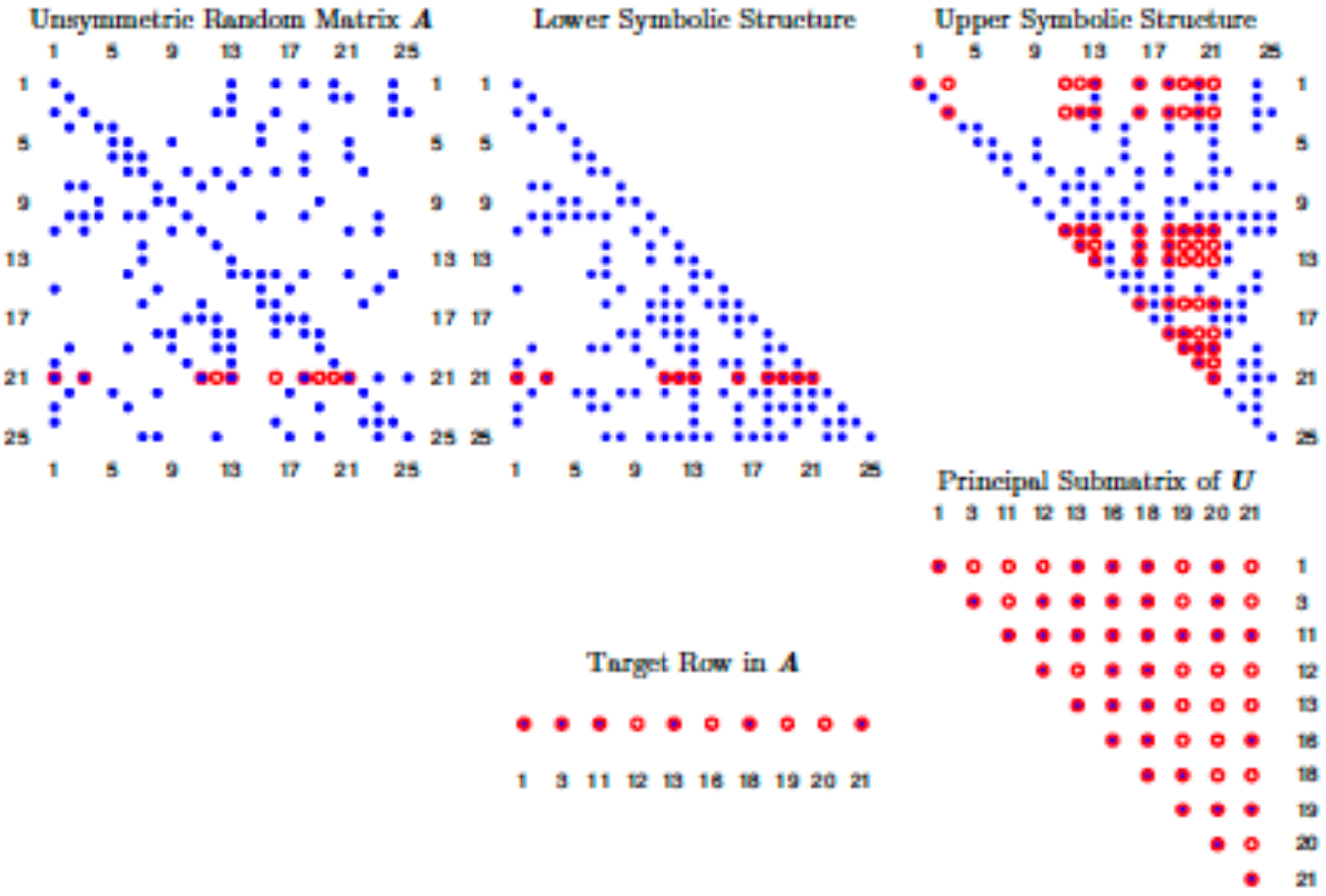


Fig. 6.1: **Illustration of entries accessed in solving for row 21 of L.** This figure shows the structure of a symbolic ILU(1) factorization applied to a randomly generated unsymmetric matrix (top left).

ATS-ILU: Results

Julia, 32 threads, shared-memory

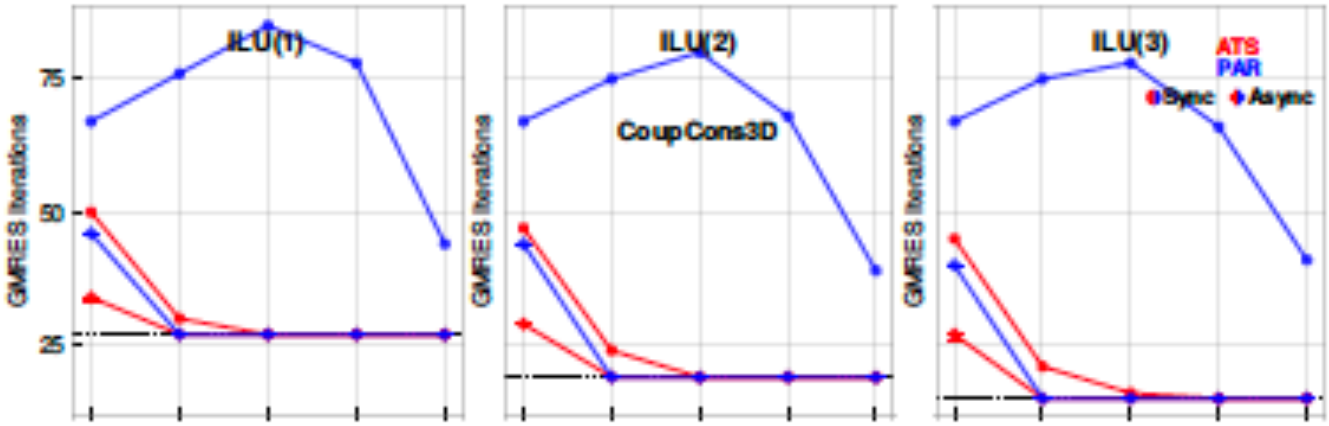
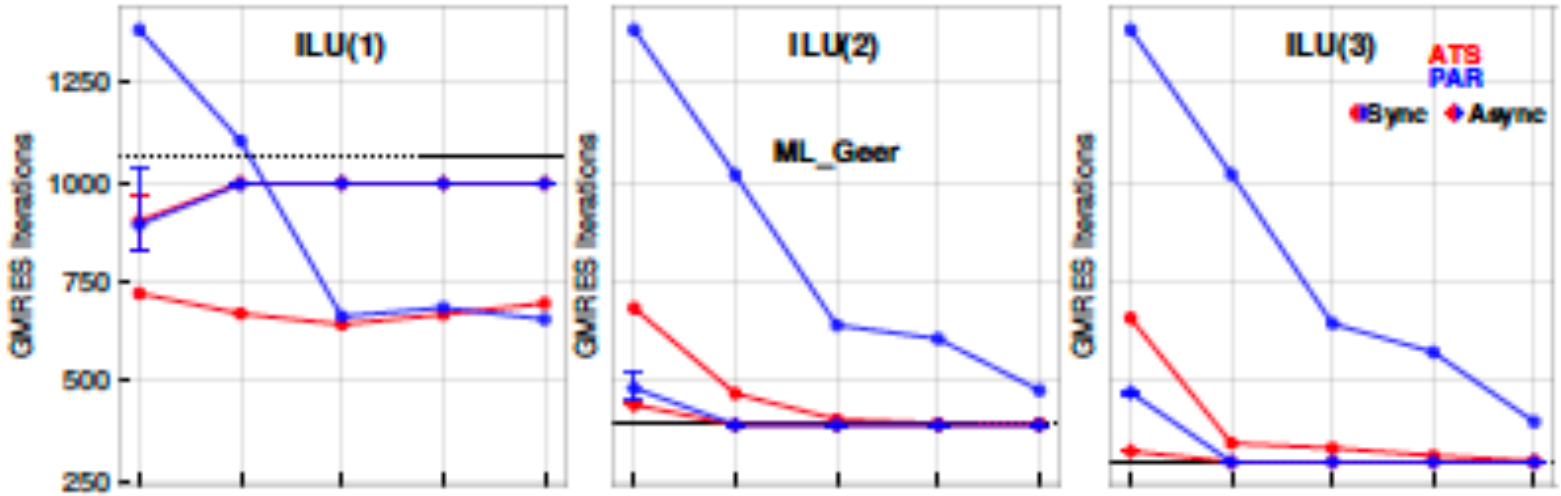


Fig. 7.1: GMRES(50) iterations vs. sweeps for *CoupCons3D* at ILU(1), ILU(2), and ILU(3). Dotted line shows the sequential ILU baseline.



ATS-ILU: SuiteSparse Results

				Synchronous						Asynchronous					
				ATS-ILU			ParILU			ATS-ILU			ParILU		
	Matrix	k	ILU(k)	1	3	5	1	3	5	1	3	5	1	3	5
Unsymmetric	CoupCons3D	1	27	50	27	27	67	85	44	54	27	27	46	27	27
	FEM_3D*	1	9	12	10	10	15	14	14	10	10	10	15	10	10
	Freescale1	2	1546	×	1405	1381	×	×	×	×	1384	1383	×	1384	1384
	ML_Geer	1	1065	720	642	695	1383	662	655	902	997	997	893	997	997
	Transport	2	1320	×	1423	1490	×	×	×	1759	1491	1491	1238	1491	1491
	atmosmodd	1	109	134	110	110	353	121	121	114	110	110	149	110	110
	atmosmodj	1	119	127	120	120	462	122	122	120	120	120	179	120	120
	atmosmodl	1	60	66	60	60	168	63	63	61	60	60	84	60	60
	atmosmodm	1	41	44	41	41	121	44	44	45	41	41	89	41	41
	crashbasis	2	×	12	7	7	28	12	8	8	7	7	13	7	7
	majorbasis	1	9	10	9	9	15	9	9	9	9	9	9	9	9
	marine1	2	1202	×	1029	1163	×	×	×	×	1103	1103	×	1103	1103
	rajat31	1	214	214	214	214	337	214	214	216	214	214	241	214	214
	ss1	1	2	2	2	2	6	3	3	3	2	2	6	2	2
	ss	1	90	132	90	90	242	92	92	93	90	90	101	90	90
	stomach	1	12	12	12	12	1037	×	×	16	12	12	42	12	12
	torso3	1	21	39	24	22	39	40	446	36	22	22	25	22	22
Symmetric	G3_circuit	1	1331	1772	1391	1409	1949	1355	1410	1410	1406	1406	1368	1406	1406
	Geo_1438	1	236	348	232	224	617	348	272	333	236	236	321	236	236
	Hook_1498	2	1143	×	1375	1156	×	×	×	1124	1143	1143	1096	1143	1143
	af_shell3	1	890	1343	826	810	×	×	×	844	800	800	832	800	800
	apache2	1	1302	×	1228	1038	×	×	1962	1599	1034	1034	1868	1034	1034
	parabolic_fem	1	559	1291	589	534	1798	945	686	562	527	527	549	527	527
	thermal1	1	1063	×	1203	1070	×	1589	1270	1245	1063	1063	1228	1063	1063

ATS-ILU: Work in Progress

GPU implementation using Kokkos
- Algorithm is GPU-friendly
- Suitable for 2-level parallelism (one team per row/column)

Inexact sparse triangular solves
- A few SPMV may be sufficient? (Neumann series)

Adaptive pattern version (ATS-ILUT)
- More expensive, but also more robust

Asynchronous Programming

How to program asynchronously?

Shared memory:

- OpenMP (CPU), CUDA (GPU)
- Kokkos (write once, run anywhere)
- Some things are still hard!

Distributed memory:

- 1-sided MPI (MPI_Put, MPI_Get)
- 2-sided MPI (JACK/JACK2)

Kokkos is (mostly) asynchronous

Kokkos supports common motifs like

- parallel for
- parallel reduce

These are by default non-blocking (“asynchronous”)

The parallel work is dispatched and executed in any order, while the main control returns early and continues to the next statement (possibly before finishing the parallel work).

Kokkos also supports streams (via execution spaces).

Thus, Kokkos is a useful tool for asynchronous algorithms.

Conclusions

Asynchronous methods hold great promise

- Asynchronous parallel execution is fine as long as the results do not change!
- Truly asynchronous algorithms are problematic because results are not reproducable
- Distributed memory: Saves idle time
- Shared memory: Increasingly important with growing #threads (e.g., GPU)
- ATS-ILU is an effective iterative ILU method
  - GPU (Kokkos) implementation in progress

Thanks to my collaborators:

Edmond Chow, Christian Glusa, Siva Rajamanickam, Daniel Szyld, Marc Tunnell, Ichi Yamazaki