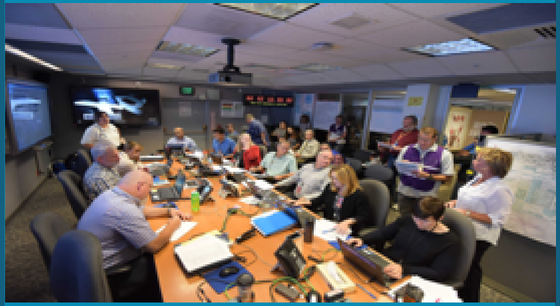
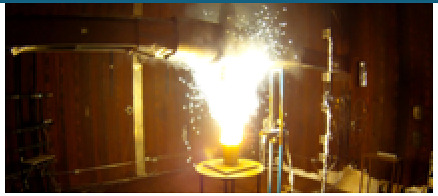
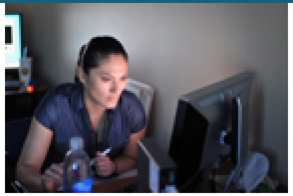
SAND2022-7459C

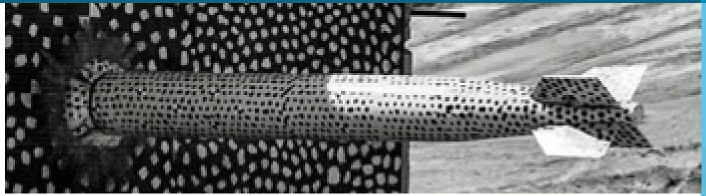
Sandia National Laboratories

Polynomial Preconditioning GMRES with Mixed Precisions









Jennifer Loe, Christian Glusa, Ichitaro Yamazaki, Erik Boman, Sivasankaran Rajamanickam, Ron Morgan

Preconditioning 2022

ECP
EXASCALE COMPUTING PROJECT

U.S. DEPARTMENT OF ENERGY
NNSA National Nuclear Security Administration

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.

SAND2022-2093 C

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.

The idea behind GMRES the (Generalized Minimum RESidual Method):

To solve $Ax = b$, where A is $n \times n$:

1. Build an orthonormal basis for a Krylov subspace:

$$\text{span}\{b, Ab, A^2b, \ldots, A^{m-1}b\}$$

2. Use an orthogonal projection to find an approximate solution which minimizes the residual:

$$\| b - Ax \|_2$$

GMRES (Generalized Minimum RESidual) Algorithm:

Algorithm GMRES (Modified Gram-Schmidt)

$\gamma = \|b\|_2$ and $v_1 = b/\gamma$
for $j = 1 : m$ **do**
    $w_j = Av_j$
    for $i = 1 : j$ **do**
        $h_{ij} = v_i^T w_j$
        $w_j = w_j - h_{ij} v_i$
    end for
    $h_{j+1,j} = \|w_j\|_2$.
    $v_{j+1} = w_j / h_{j+1,j}$
end for
Define the $(m+1) \times m$ matrix $\overline{H} = \{h_{ij}\}$
Solve least-squares problem $\overline{H}d = \gamma e_1$ for d.
$\hat{x} = V_m d$

Sparse Matrix-Vector Product (SpMV)

Orthogonalizing the next basis vector

Restart when subspace size gets too large!

See details in "Iterative Methods for Sparse Linear Systems 2nd ed." by Saad.

Implementing the Polynomial Preconditioner

$$
\begin{aligned}
Ap(A)y &= b, \\
x &= p(A)y.
\end{aligned}
$$

$$Ap(A) = I - \prod_{i=1}^{d}\left(I - \frac{1}{\theta_i}A\right)$$

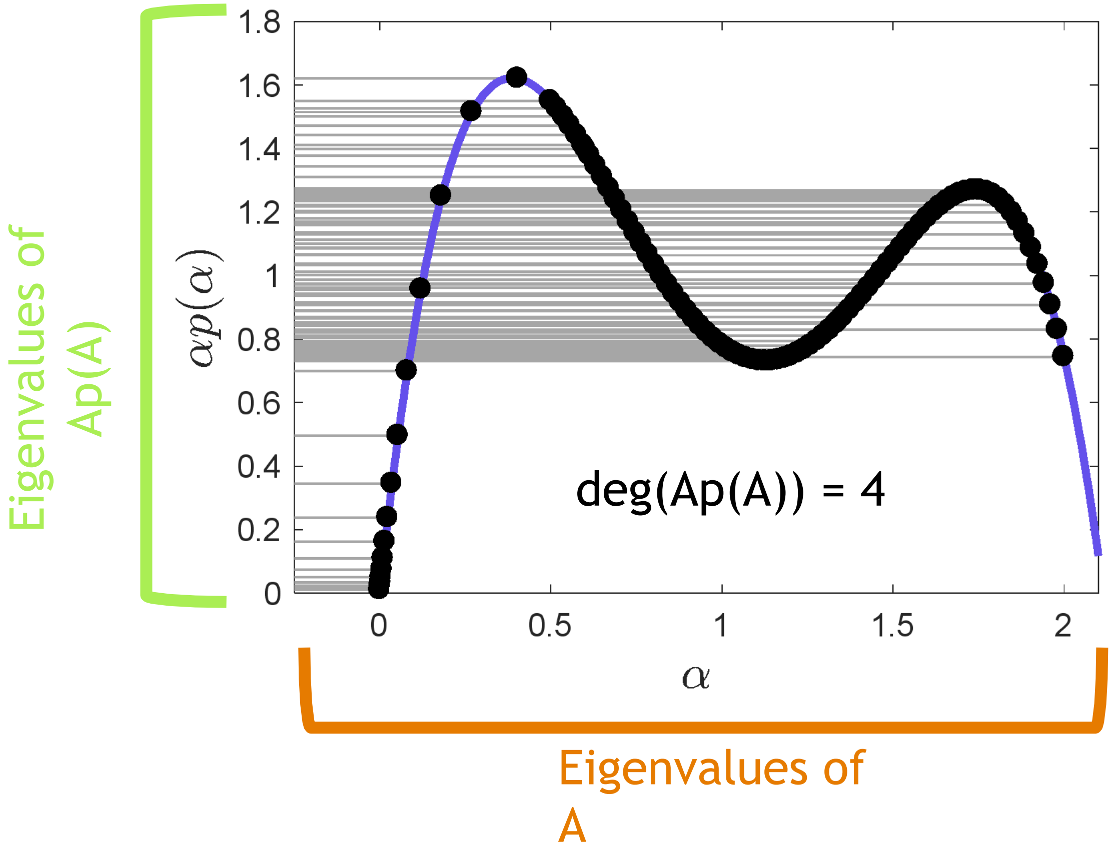
$$p(A) = \sum_{k=1}^{d}\frac{1}{\theta_k}\left(I - \frac{1}{\theta_1}A\right)\left(I - \frac{1}{\theta_2}A\right)\cdots\left(I - \frac{1}{\theta_{k-1}}A\right)$$

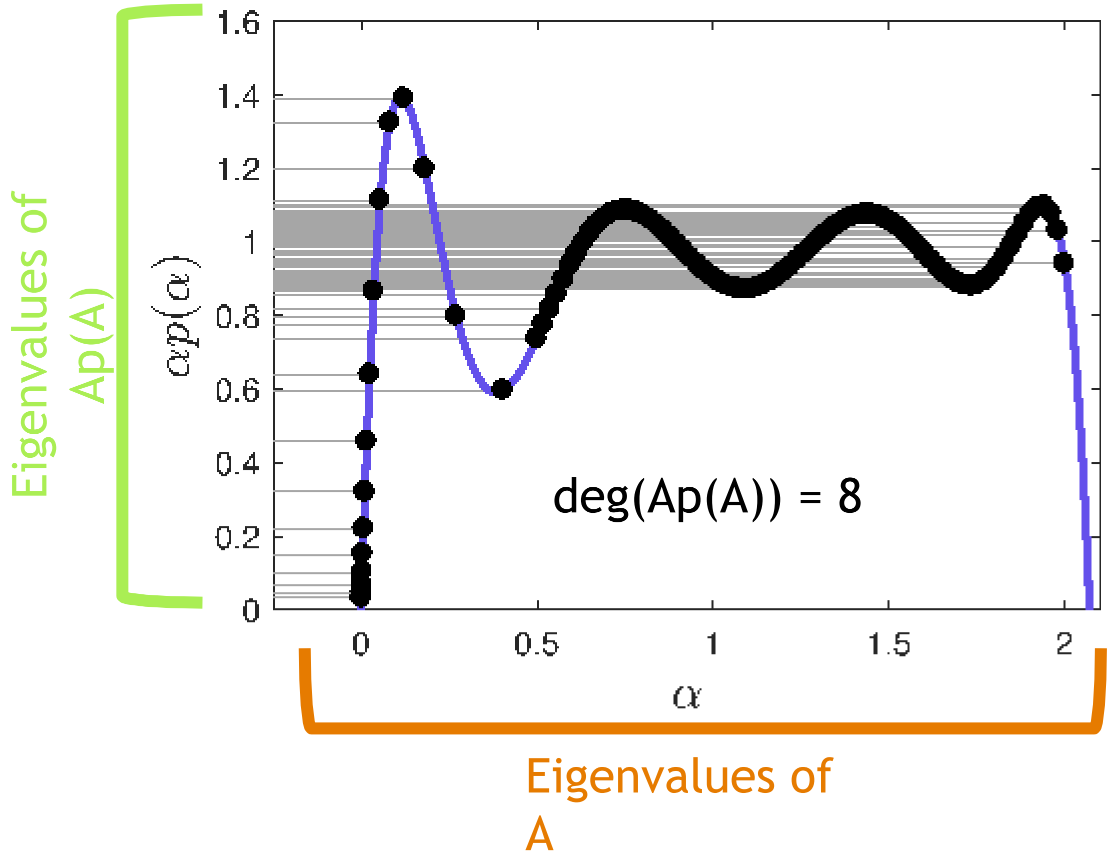
[See: Toward Efficient Polynomial Preconditioning for GMRES, J. Loe and R. Morgan, *Numerical Linear Algebra with Applications*, December 2021.]

Generating the polynomial preconditioner: p(A) has degree d-1; Ap(A) has degree d

1. Run one cycle of GMRES(d) using a random starting vector.
2. Find the harmonic Ritz values $\theta_1, \ldots, \theta_d$, which are the roots of the GMRES polynomial:
   With Arnoldi decomposition $AV_d = V_{d+1}H_{d+1,d}$, find the eigenvalues of $H_{d,d} + h^2_{d+1,d} f e_d^T$, where $f = H_{d,d}^{-*} e_d$ with elementary coordinate vector $e_d = [0, \ldots, 0, 1]^T$.
3. Order the GMRES roots with *modified Leja ordering* [Bai, Hu, Reichel]
   (This ordering uses products of absolute values of differences of roots.)

Remapping Eigenvalues (Symmetric Matrix)





Polynomial Preconditioning to Accelerate ILU:

- Compose polynomial preconditioning with other preconditi
- Matrix III Stokes.
- ILU(0.001) is computed from the shifted matrix A + 0.001I

$$AMp(AM)y = b,$$

$$x = Mp(AM)y.$$

degree d	cycles	*mvps*	*vops*	dot products	time
No Standard Preconditioning					
1	485,042	$2.43 * 10^7$	$1.36 * 10^9$	$6.43 * 10^8$	21.6 hours
50 + 5	1072	$2.95 * 10^6$	$5.90 * 10^6$	$1.42 * 10^6$	29.9 min's
100 + 20	277	$1.66 * 10^6$	$2.43 * 10^6$	$3.71 * 10^5$	15.2 min's
With ILU Preconditioning					
1	958	47,902	$2.69 * 10^6$	$1.27 * 10^6$	211 sec's
50	3	7051	16,978	4799	13.6 sec's
100 + 10	2	7691	21,273	6668	14.8 sec's

Why incorporate lower precisions in GMRES?

- Reduce data movement to overcome memory-bound algorithms.
- Use cheaper floating-point operations.

Obstacles to lower precision:

- Lower precision computations result in more roundoff error!
- …but applications still need high level of accuracy in solutions.
- Tricky to find where to use lower precision in algorithm while maintaining accuracy.

So how DO we use lower precision in GMRES?

Iterative Refinement with GMRES (GMRES-IR)

Algorithm 1 Iterative Refinement with GMRES Error Correction

$r_0 = b - Ax_0$ [double]
for $i = 1, 2, \ldots$ until convergence: **do**
  Use GMRES(m) to solve $Au_i = r_i$ for correction u_i [single]
  $x_{i+1} = x_i + u_i$ [double]
  $r_{i+1} = b - Ax_{i+1}$ [double]
end for

(At each restart, update solution vector and recompute residuals in double precision.)

Note: We store TWO copies of matrix A (double and single).

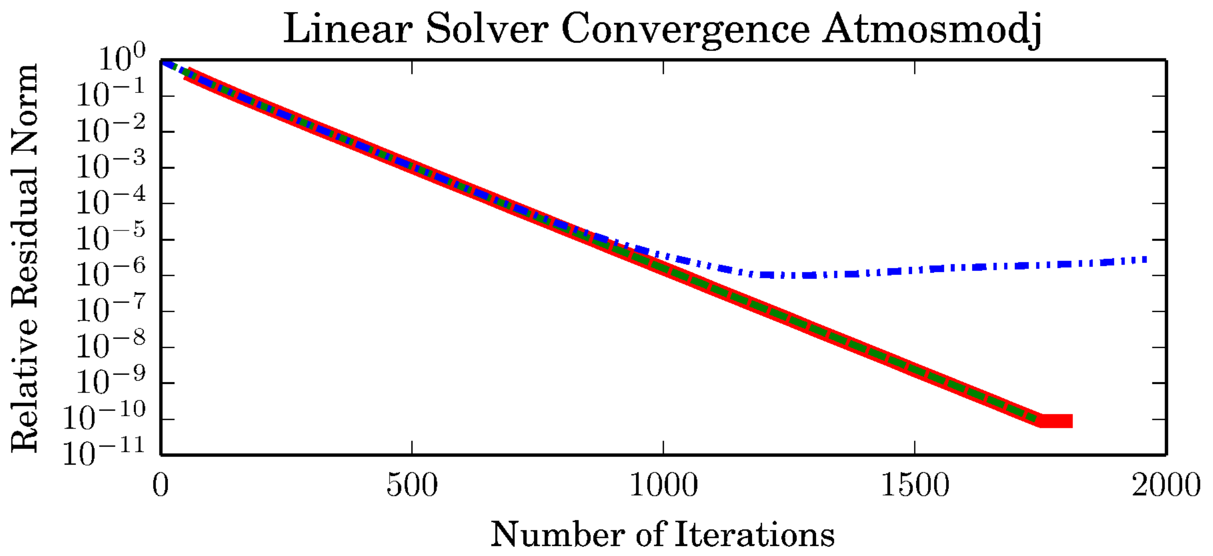
Not a new algorithm. See related works:

- Neil Lindquist, Piotr Luszczek, and Jack Dongarra. *Improving the performance of the GMRES method using mixed-precision techniques.*
- Hartwig Anzt, Vincent Heuveline, and Bjorn Rocker. *Mixed precision iterative refinement methods for linear systems: Convergence analysis based on Krylov subspace methods.*
- Erin Carson and Nicholas J. Higham. *Accelerating the solution of linear systems by iterative refinement in three precisions.*

How does convergence of GMRES-IR compare to GMRES double?

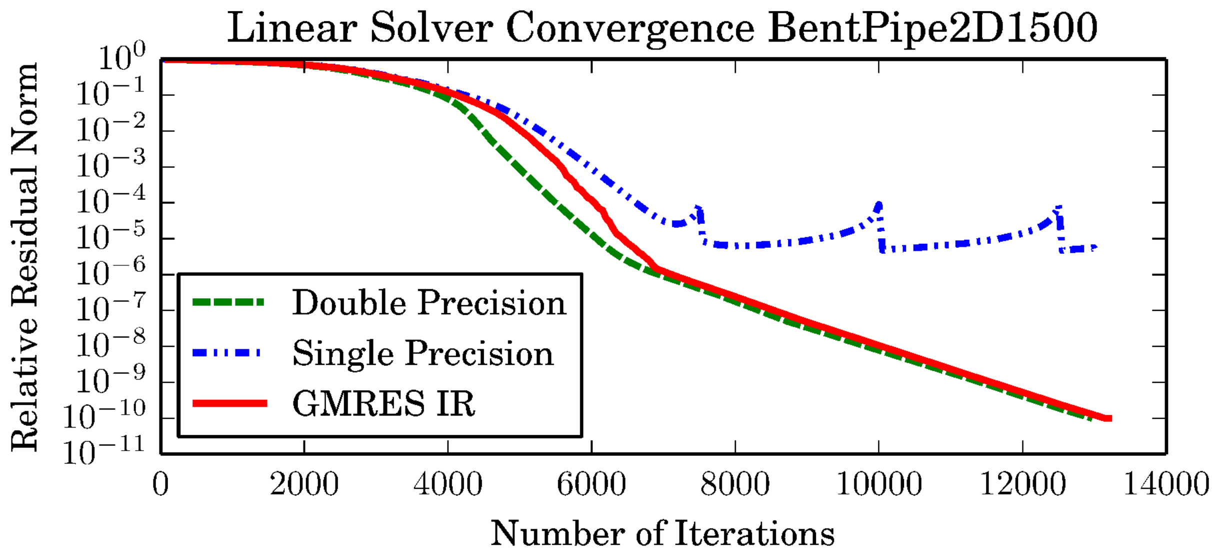
Atmosmodj:

- SuiteSparse, cfd
- n = 1,270,432
- GMRES Double: 5.12s, 1740 iterations
- GMRES-IR: 3.78s, 1750 iterations



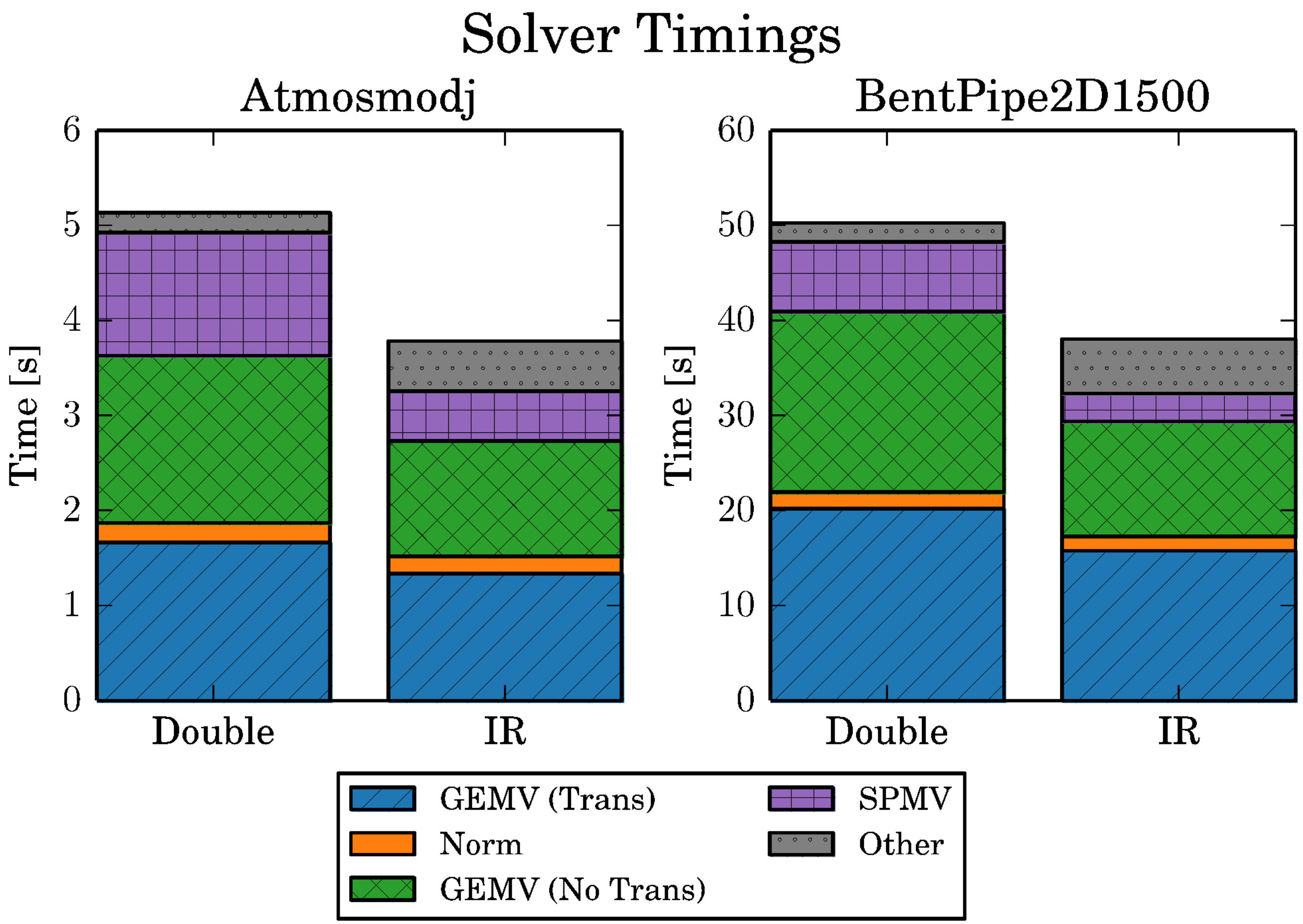
BentPipe2D1500:

- 2D convection-diffusion
- n = 2.25 million
- GMRES Double: 50.26s, 12,967 iterations
- GMRES-IR: 38.03s, 13,150 iterations



GMRES-IR convergence follows convergence of GMRES Double!

Kernel Speedup:



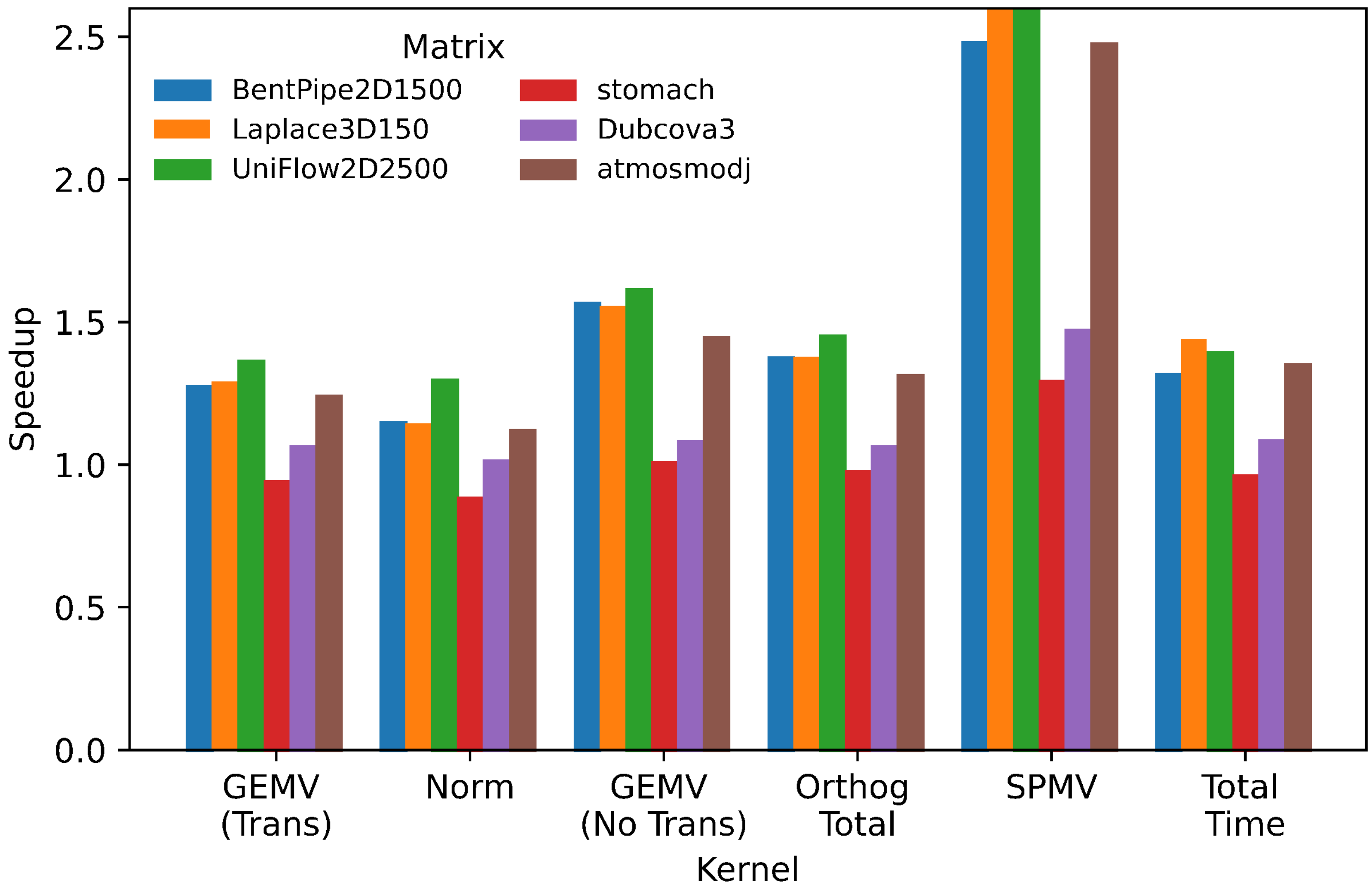
Atmosmodj:

- GMRES Double: 5.12s, 1740 iterations
- GMRES-IR: 3.78s, 1750 iterations

BentPipe2D1500:

- GMRES Double: 50.26s, 12,967 iterations
- GMRES-IR: 38.03s, 13,150 iterations

Kernel speedups compared with other matrices:



A model for L2 cache use with low precision SpMV:

Suppose that A has w nonzero elements per row and n rows (so $nnz = w*n$).

A stored in CSR format with 2 vectors of size $w*n$:

Values of A: A_{val} Column indices: $colId$ (Ignore vector of row ptrs)

Computing the first dot product of the SPMV:

$$\sum_{i=0}^{w-1} A_{val}[i] * x[colId[i]].$$

Case: fp64 with no cache reuse (i.e. every element of x has to be read into cache every time needed):

$$n * w * [size(int) + 2 * size(double)] = 20wn.$$

Case: fp32 with "perfect" cache reuse (i.e. any elements of x read into cache stay in cache until not needed):

$$n*w*[size(int)+size(float)]+n*size(float) = (8w+4)n.$$

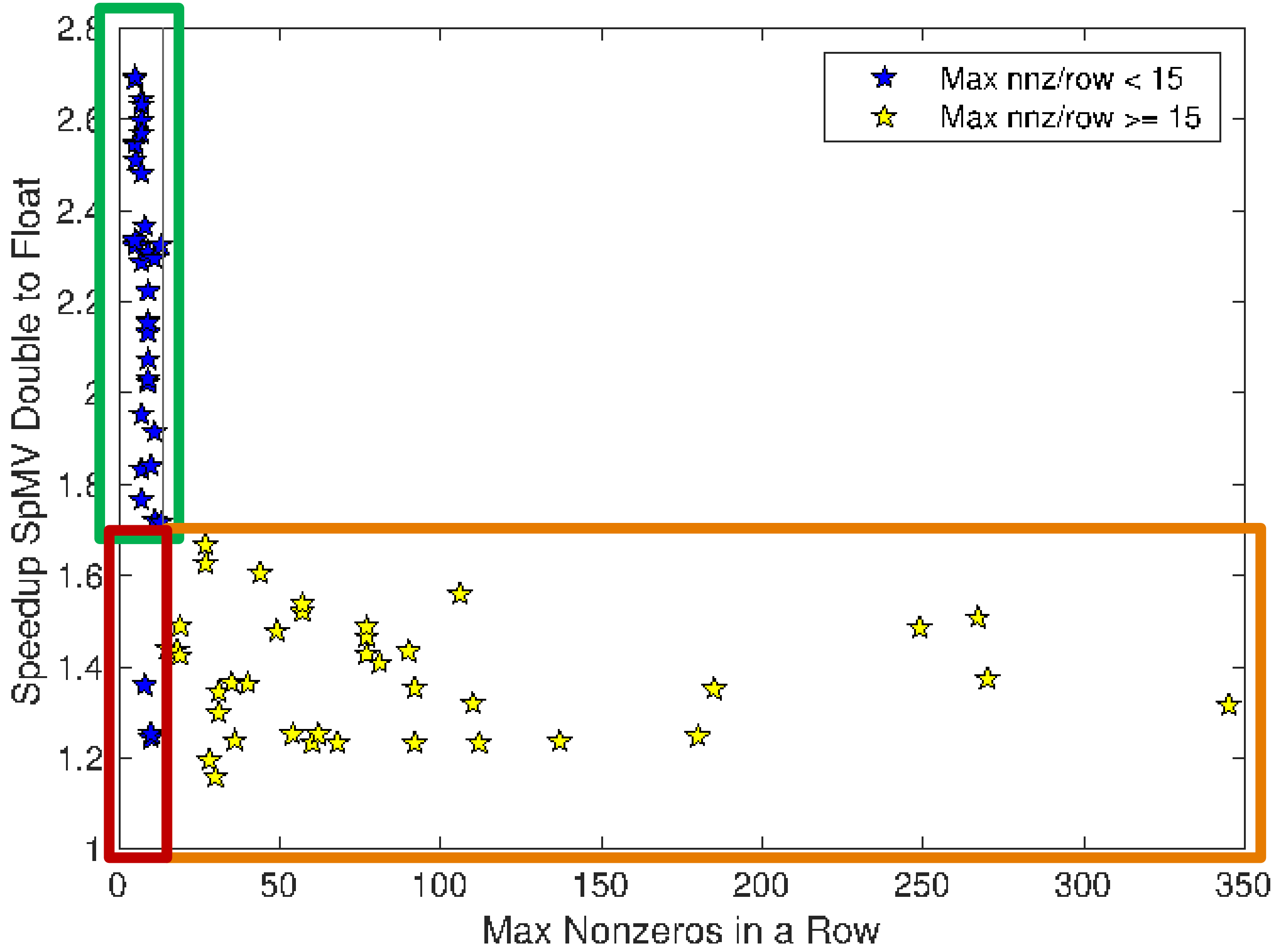
Expected speedup: $\dfrac{20wn}{(8w+4)n} = \dfrac{5w}{2w+1}.$ → 2.5 as w gets large.

** Thanks to Christian Trott and Luc-Berger Vergiat for help in creating this model!

SpMV Speedup vs Nonzero Structure of Matrix:

Very good speedup for matrices w/ small nnz/row.

Three smallest matrices in test set.

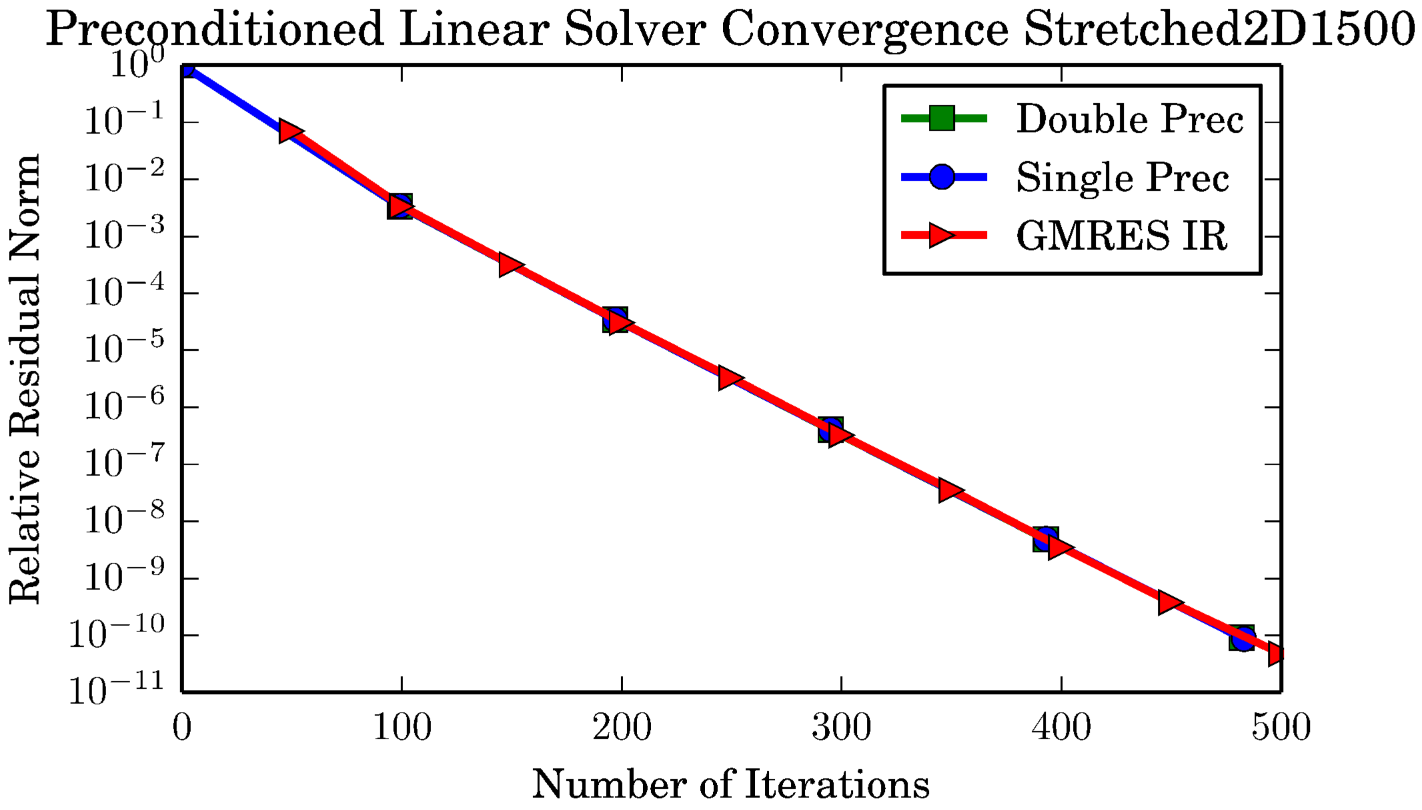


Large max nonzeros per row; low SpMV speedup

How does preconditioning affect GMRES-IR convergence?

Stretched2D1500:
- 2D Laplacian on Stretched Grid
- n = 2.25 million

Polynomial Preconditioner:
- GMRES Polynomial
- GMRES double:
  double precision poly preconditioner
- GMRES-IR:
  single precision poly preconditioner

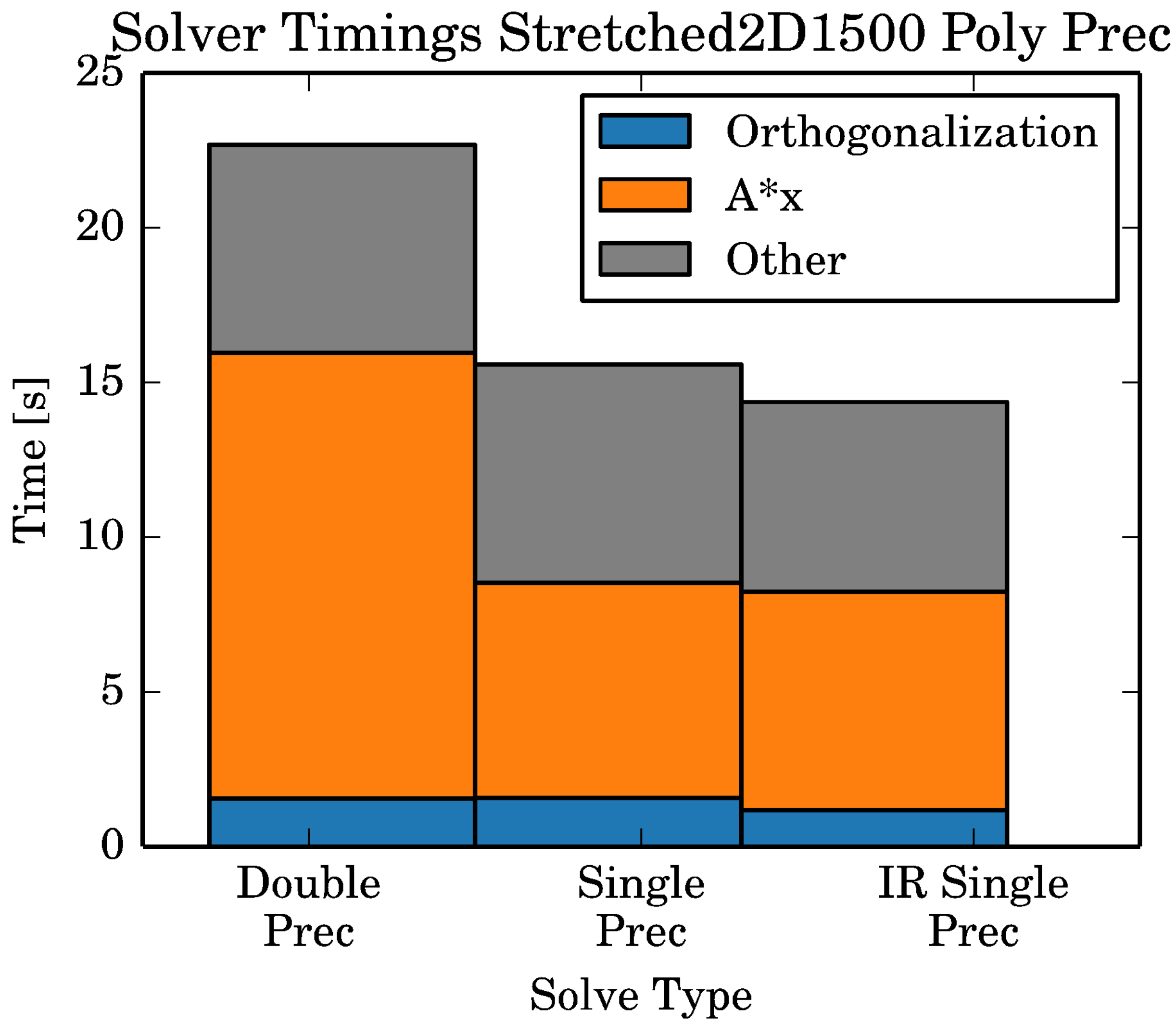


Preconditioned GMRES-IR convergence still follows convergence of GMRES Double!

Polynomial Preconditioning

LEFT: GMRES double w/ **fp64** polynomial preconditioner.
MIDDLE: GMRES double w/ **fp32** polynomial preconditioner.
RIGHT: GMRES-IR w/ **fp32** polynomial preconditioner.

Polynomial preconditioning shifts main expense to SpMV rather than dense orthogonalization kernels.



**For polynomial preconditioning details, see:
Jennifer Loe, Erik Boman, and Heidi Thornquist. *Polynomial Preconditioned GMRES in Trilinos: Practical Considerations for High-Performance Computing*

Results from SuiteSparse Matrices:

				Double		IR		
UF id	Matrix Name	N	prec	Time	Iters	Time	Iters	Speedup
2266	atmosmodj	1,270,432		5.12	1740	3.78	1750	1.35
2267	atmosmodl	1,489,752		1.61	446	1.23	450	1.31
1858	crashbasis	160,000		0.55	431	0.52	450	1.07
1849	Dubcova3	146,698		1.15	1131	1.05	1150	1.10
1852	FEM_3D_thermal2	147,900		0.84	775	0.80	800	1.05
1853	parabolic_fem	525,825		42.39	27493	44.63	36600	0.95
1367	SiO2	155,331		18.23	17385	16.86	17600	1.08
895	stomach	213,360		0.51	359	0.52	400	0.98
2259	thermomech_dM	204,316		0.27	88	0.27	100	1.00
894	lung2	109,460	j 1	0.46	206	0.49	250	0.94
1266	hood	220,542	j 42	13.98	5762	9.04	5000	1.55
805	cfd2	123,440	p 25	6.05	1092	4.55	1100	1.33
1431	filter3D	106,437	p 25	25.24	4449	18.12	4450	1.39
2649	Transport	1,602,111	p 25	8.35	339	8.73	450	0.96
	BentPipe2D1500	2,250,000		50.26	12967	38.03	13150	1.32
	Laplace3D150	3,375,000		16.93	2387	11.75	2400	1.44
	UniFlow2D2500	6,250,000		29.62	2905	21.17	3000	1.40
	Stretched2D1500	2,250,000	p 40	22.66	482	14.37	500	1.58

*prec column:
p = polynomial prec w/ degree
j = Jacobi prec w/ block size

Blue: Block Jacobi Preconditioned

Green: Polynomial Preconditioned

Future Work:

- Implement GMRES-IR in Tpetra solvers in Belos package of Trilinos
- Make GMRES (double) with single precision preconditioning available in Tpetra Belos solvers.
- Incorporate half precision computations (fp16 and bfloat16).
- Test performance on other (non-NVIDIA) GPU architectures- AMD and Intel.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.