SAND2019-7457C

Polynomial Preconditioning for Avoiding Communication in GMRES

Jennifer Loe, Heidi Thornquist, Erik Boman

Baylor University, Sandia National Laboratories

July 1, 2019

Polynomial Preconditioning:

Solving Polynomial Preconditioned System: $Ax = b$ becomes

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

where $p(A)$ is a polynomial of degree d.

Choose p to be the minimum residual polynomial from GMRES.

Old Krylov Subspace:

$$\mathcal{K} = span\{b, Ab, A^2 b, \ldots, A^{m-1} b\}$$

New Krylov Subspace:

$$\mathcal{K} = span\{b, Ap(A)b, (Ap(A))^2 b, \ldots, (Ap(A))^{m-1} b\}$$

Why precondition with the GMRES polynomial??

- Reduce number of GMRES iterations.
- More work done between orthogonalization steps! Avoid global synchronous communication!
- Often also reduces number of matrix-vector products.
- General-purpose preconditioner. Straightforward to compute.
- Can be combined with other preconditioners!
- It's available in Trilinos! (almost)

Obtaining the polynomial:

To find the polynomial p of degree d:

1. Run d steps of GMRES on the matrix A, using a random right-hand side.
   (To combine with another preconditioner M, run d steps of GMRES on AM.)
2. Use the resulting matrices to compute the harmonic Ritz values θ_i of A. (or AM.)
3. Order the θ_i's using a Modified Leja ordering.
4. Use the θ_i's to apply the polynomial as a preconditioner.

(Also options for root-adding or damping if needed for stability.)
[See Embree, Loe, Morgan 2018]

Polynomial Preconditioning: Implementation

Solving Preconditioned System:

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

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

$$p(A) = \sum_{k=1}^{d} \frac{1}{\theta_k} \left(1 - \frac{1}{\theta_1} A\right)\left(1 - \frac{1}{\theta_2} A\right) \cdots \left(1 - \frac{1}{\theta_{k-1}} A\right) \tag{2}$$

The θ_i's are Harmonic Ritz values, a byproduct of GMRES.

Polynomial Preconditioning: Implementation

Option 1: Use both formulas. (See previous presentation.)

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

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

$$p(A) = \sum_{k=1}^{d} \frac{1}{\theta_k} \left(1 - \frac{1}{\theta_1} A\right) \left(1 - \frac{1}{\theta_2} A\right) \cdots \left(1 - \frac{1}{\theta_{k-1}} A\right) \tag{2}$$

Advantage: Simpler formula. Less vector additions.
Disadvantage: Possible stability issues applying different operator.

Polynomial Preconditioning: Implementation

Option 2: Use one formula. (Implemented in Trilinos.)

$$Ap(A)y = b,$$
$$x = p(A)y.$$

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

$$p(A) = \sum_{k=1}^{d} \frac{1}{\theta_k} \left(1 - \frac{1}{\theta_1} A\right) \left(1 - \frac{1}{\theta_2} A\right) \cdots \left(1 - \frac{1}{\theta_{k-1}} A\right) \qquad (2)$$

Advantage: Applying a consistent operator.
Disadvantage: Up to 2x as many vector additions.

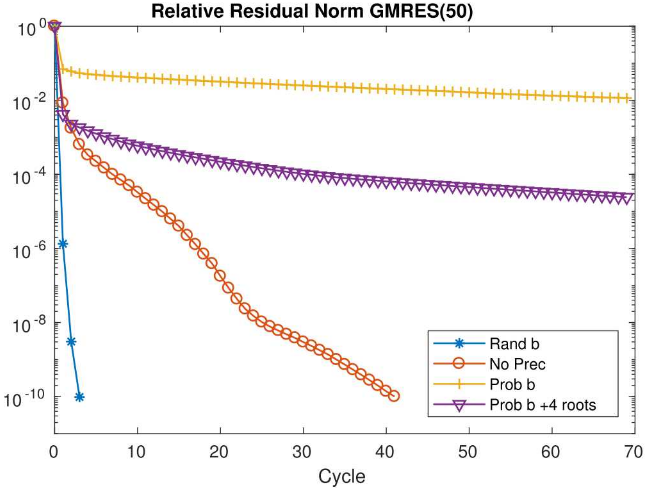
Vector choice for polynomial generation:

Q: Why run GMRES with a random right-hand side to generate the polynomial??

Q: Why not use the problem right-hand side and get an initial guess for preconditioned GMRES?

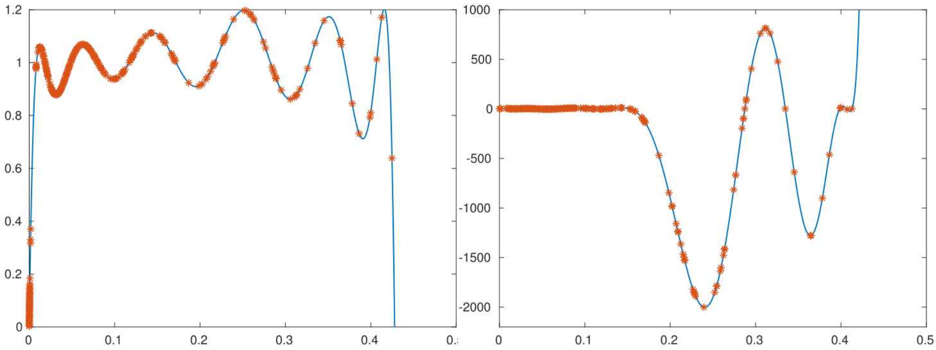
A: Not a great idea... use at your own risk!

Vector choice for polynomial generation:



Convergence for the Memplus circuit matrix using different degree 15 polynomials. [From experiments run in Matlab with Ron Morgan.]

Vector choice for polynomial generation:



(a) Polynomial generated with b_{rand} (b) Polynomial generated with b_{prob}

Figure: Polynomials of degree 15 plotted over the real axis on $[0, 0.5]$. Stars indicate eigenvalues of the Memplus matrix.

[From experiments run in Matlab with Ron Morgan.]

Vector choice for polynomial generation:

Precautions for using random vectors:

- The polynomial will change as we increase MPI processes!
- Bad random vector generator $\Longrightarrow$ bad polynomial.

Trilinos

Belos: Iterative Linear Solvers Package: CG, GMRES, Block Krylov methods, BiCGStab
Other Capabilities: Algebraic preconditioners (IFPACK), load partitioning (Zoltan), Direct Solvers (Amesos), Multigrid (MueLu), Eigensolvers (Anasazi), ...
Application Areas: Circuit simulation, Ice sheet modeling, hydrodynamics, geophysics, ...

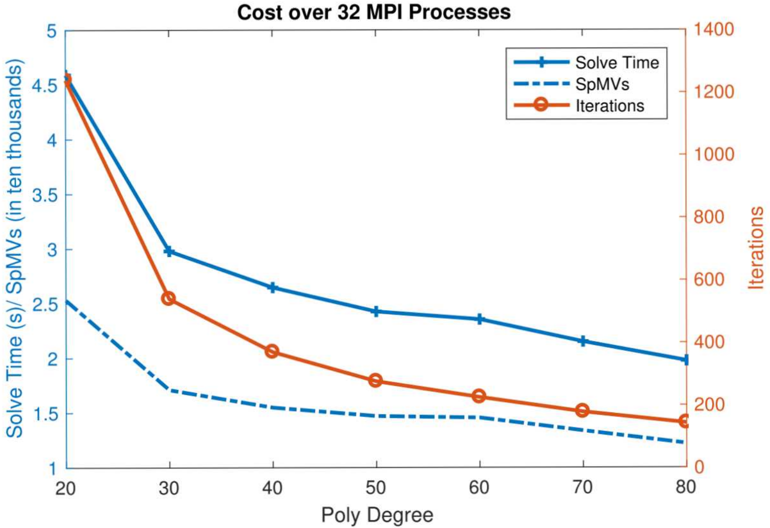
Polynomial Preconditioning Options in Trilinos

Implemented in Belos.
Include header file: "BelosGmresPolySolMgr.hpp"

- **poly-type** - "Roots" to get cheapest/most stable version
- **max-degree** - maximum polynomial degree
- **orthogonalization** - orthogonaliation in GMRES to create polynomial (We use ICGS.)
- **use problem rhs** - Default OFF. (For reasons discussed above.)
- **add roots** - Default ON for stability
- **damp** - Default OFF. Sometimes useful for indefinite problems.
- **outer solver** - ANY solver in Belos' solver factory!
- **outer solver params** - parameter list to pass to the outer solver

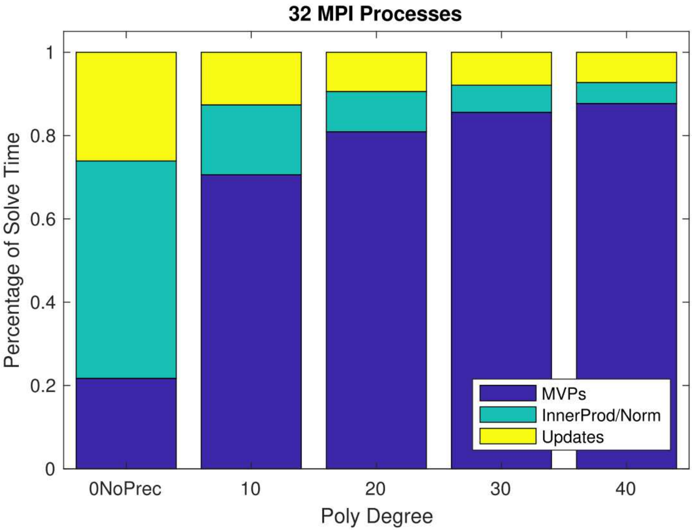
A Small CFD example:

Matrix cfd2, A is SPD, $n = 123440$. GMRES(50) with b random.



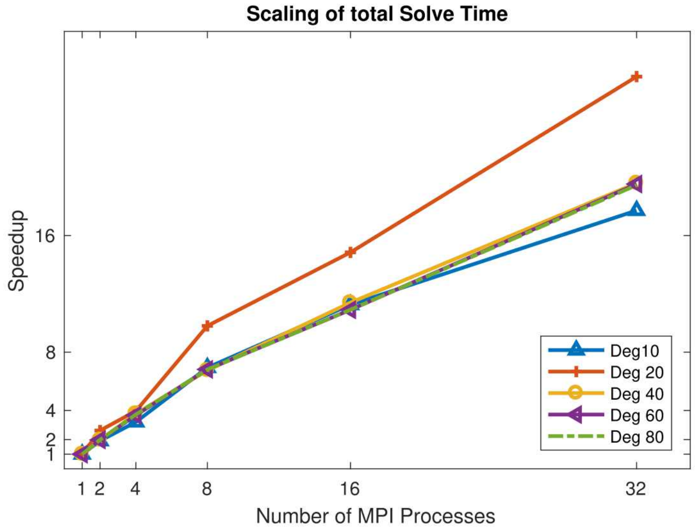
A Small CFD example:

Comparing solve time distribution per polynomial degree:



A Small CFD example:

Strong Scaling:



Combining with other Preconditioners

Poly preconditioning alone:

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

With other preconditioners:

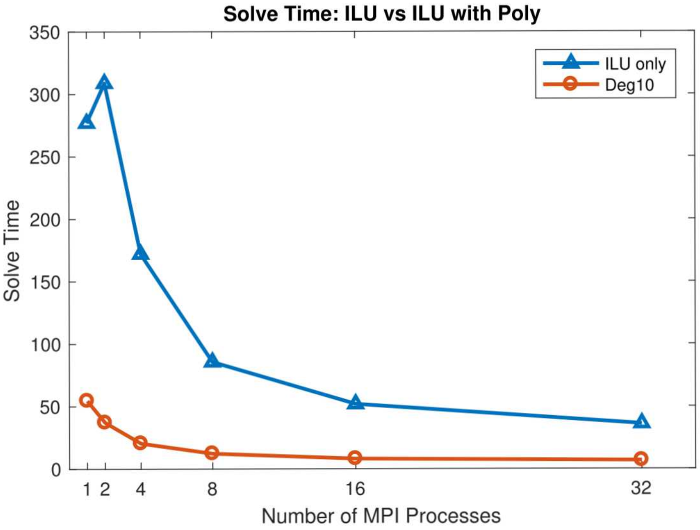
$$\begin{aligned} AMp(AM)y &= b, \\ x &= p(AM)y. \end{aligned}$$

No extra work to code this in your Trilinos solver!
Just pass your preconditioner M to the linear problem like usual.

New Example with ILU:

- Matrix: Transport (From SuiteSparse Janna collection)
- Problem: 3D finite element flow and transport
- Size: $n = 1{,}602{,}111$
- Nonzeros: 23,487,281
- Non-symmetric
- ILU Fill: 1
- ILU overlap: 0
- Load balancing: Zoltan hypergraph partitioning
- GMRES(100)
- rtol: $1e-8$

Solve time: ILU vs ILU with Poly

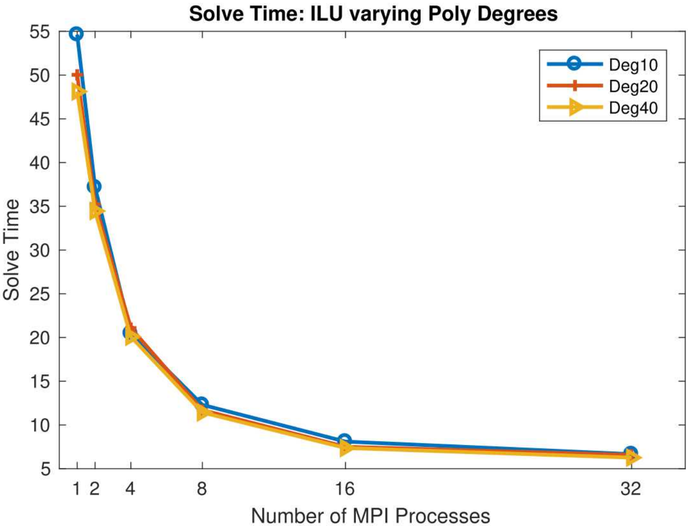


Solve time: ILU vs ILU with Poly

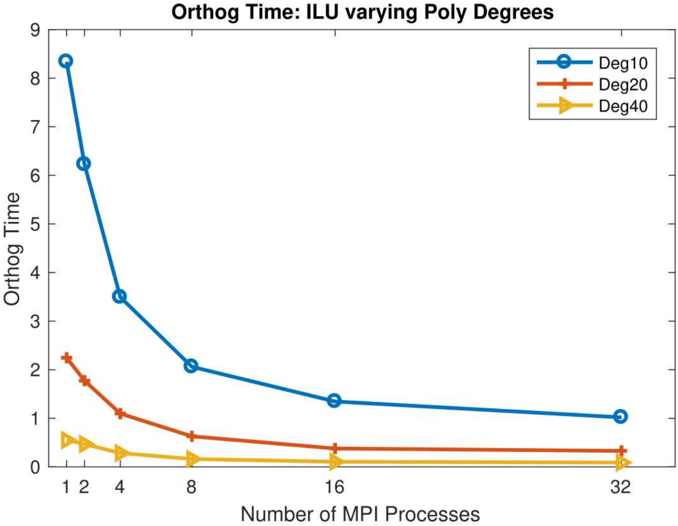
MPI Procs	ILU	ILU+Deg 10	Improvement
32	36.05	6.67	5.07x
16	51.91	8.10	8.31x
8	85.67	12.30	8.42x
4	172.1	20.45	6.97x
2	308.8	37.16	6.41x
1	277	54.62	5.41x

ILU has fill level of 1 and no overlap.

Solve time: ILU with different Poly Degreees



Comparing Orthogonalization Time:



Degree 10 poly spends more than 10x as much time in orthogonalization as degree 40 poly.

Preconditioner Generation Time

Over 32 MPI processes.
(Solve time does not include preconditioner setup time.)

	Prec Setup Time	**Solve Time**
ILU	0.2157	36.05
ILU + Deg 10	0.3334	6.66
ILU + Deg 20	0.4922	6.58
ILU + Deg 40	0.8659	6.27
Deg 20	0.1754	26.16
Deg 40	0.4926	15.19
Deg 80	1.655	14.62

Communication Avoiding S-Step GMRES

Delayed Orthogonalziation:
Can avoid dot products in GMRES by orthogonalizing every s steps:
E.g. $s = 3$:

$$\mathcal{K} = span\{b, Ab, A^2 b, A^3 b, A^4 b, A^5 b, A^6 b, \ldots, A^{m-1} b\}$$

Use TSQR to orthogonalize the blocks.

Matrix Powers Kernel:

- Used for performing repeated matvecs with A.
- Minimizes the number of reads from slow memory and cache.

[Demmel, Hoemmen, et al.]

Combining with Communication-Avoiding Methods

1. **Polynomial preconditioned standard GMRES.**
   - Use Matrix Powers Kernel (MPK) to evaluate the polynomial.
   - Take advantage of CA kernels while avoiding pitfalls of delayed orthogonalization. (Can use MPK without worrying with Newton basis, etc.)
2. **Polynomial preconditioning within CA-GMRES.**
   - More SpMVs per orthogonalization.

Future Work:

- Large-scale experiments
- More applications (Circuit matrices?)
- Direct comparison with CA-GMRES?
- Implement in combination with communication-avoiding kernels?

Thank you!

This research was funded in part by Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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-NA-0003525. This work was in part supported by the Department of Energy's Exascale Computing Project (ECP).