Polynomial Preconditioned GMRES in Trilinos: Practical Considerations for High-Performance Computing*

Jennifer A. Loe† Heidi K. Thornquist‡ Erik G. Boman§

Abstract

Polynomial preconditioners for GMRES and other Krylov solvers are well-known but are infrequently used in large-scale software libraries or applications. This may be due to stability problems or complicated algorithms. We implement the GMRES polynomial as a preconditioner in the software library Trilinos and demonstrate that it is stable and effective for parallel computing. Trade-offs when selecting a polynomial degree and combining with other preconditioners are analyzed. We also discuss communication-avoiding (CA) properties of the polynomial and relate these to current CA-GMRES methods.

1 Introduction

The computational simulation of physics-based models often requires solves of large, sparse linear systems $Ax = b$. Due to storage requirements and computational complexity, Krylov solvers such as CG, TFQMR, BiCGStab, and the Generalized Minimum Residual method (GMRES) [24] are commonly chosen over direct solvers. As we work to support these simulations at exascale, communication has become a bottleneck for these solvers while floating point operations are comparatively inexpensive. A Krylov solver builds out a basis for a Krylov subspace $\mathcal{K}_m(A, b) = span\{b, Ab, A^2b, \ldots, A^{m-1}b\}$ from which to extract an approximate solution vector $\hat{x}$. The dot products required for orthogonalizing the basis vectors are especially costly as they require global all-to-all communication and serve as a synchronization point. GMRES in particular suffers the effects of dot products since each new basis vector must be fully orthogonalized against previous basis vectors. While classical Gram-Schmidt allows some dot products to be combined into a single block dot product, the cost of communication from dot products can still be a bottleneck at large scale.

Krylov solvers are typically used in combination with a preconditioner M to accelerate convergence. For a right-preconditioned system, one solves a linear system $AMy = b$ where $x = My$ is the solution to the original system $Ax = b$. The matrix M is chosen to be a good approximation to A^{-1} so that AM approximates the identity and has an easier spectrum for Krylov method convergence. A variety of preconditioners exists including ILU(k), Jacobi, and algebraic multigrid. Some preconditioners perform well when used with increasing levels of parallelism, but others suffer performance degradation due to domain decomposition or increased communication expense. A polynomial preconditioner has the form $M = p(A)$ where p is a polynomial. Such preconditioners have been studied extensively; see, for instance, [2, 26, 9, 10, 15, 17, 20, 21, 22]. Polynomial preconditioners are well-suited to parallel computing because they require only sparse matrix-vector products (SpMVs) and vector additions (AXPYs) to apply. AXPYs are operations of the form $\alpha x + y$, which require no communication, and SpMVs are assumed to require only local communication between neighboring processors. Thus, polynomials can help improve Krylov solver convergence without the burden of additional global communication.

While polynomial preconditioners have potential to reduce communication in parallel computing, they are not typically in the first line of defense. This may be due to concerns about stability [14] or due to added costs for computing spectral information to generate the polynomial [23, p. 395]. Furthermore, some methods for generating polynomial preconditioners are quite complicated to implement. New polynomial preconditioners are being developed for parallel applications both on CPUs [13, 16] and GPUs [12, 27, 11], but they are often limited to symmetric problems or specific applications.

The GMRES polynomial preconditioner avoids most of these difficulties. It has been studied for both linear equations [1, 18, 19] and eigenvalue problems [25, 5]. It can be applied to both symmetric and non-symmetric problems. Good eigenvalue estimates are not required to generate the polynomial. The polynomial is

*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 supported in part by the Department of Energy's Exascale Computing Project (ECP).

†Sandia National Laboratories

‡Sandia National Laboratories

§Sandia National Laboratories.

simple to construct and can be easily composed with other preconditioners. For high-degree polynomials, added roots can give stability. Even though applying a polynomial requires more SpMVs at each iteration, the GMRES polynomial preconditioner often reduces the total number of SpMVs and iterations needed to attain convergence. Thus, the polynomial helps to progress the GMRES iteration while reducing the global communication and synchronization costs of orthogonalization. In addition to its communication-avoiding properties, polynomial preconditioned (PP)-GMRES is well-suited to large-scale computing for a number of other reasons: The storage requirements for the GMRES polynomial preconditioner are minimal; only a small vector of harmonic Ritz values needs to be stored. Since direct access to the matrix is not required, the polynomial can be constructed in a matrix-free environment. Furthermore, the polynomial preconditioner helps GMRES to overcome limitations imposed by restarting. In this paper, we demonstrate that the GMRES polynomial preconditioner can be practical and effective for large-scale applications.

The contributions of this paper are as follows: We present the GMRES polynomial preconditioner that has been implemented in the software library Trilinos. We apply the polynomials to several practical problems (some in conjunction with other preconditioners) to show that the GMRES polynomial is robust, reliable, and effective. Furthermore, we demonstrate that the GMRES polynomial preconditioner can reduce othogonalization steps resulting in global communication. Throughout all experiments, we present considerations for selecting the best polynomial degree and tradeoffs for combining with other preconditioners. Finally we compare the GMRES polynomial preconditioner to other communication-avoiding (CA)-GMRES variants and discuss the advantages of possible combinations.

2 Background and Implementation

2.1 Generating and Applying the Polynomial

To compute a GMRES polynomial preconditioner $p(A)$ for the system $Ax = b$, we run d steps of GMRES using a random starting vector $\tilde{b}$. Then we compute the harmonic Ritz values, which are roots of the polynomial $I - Ap(A)$, and use them to apply $p(A)$ in our preconditioned iteration, as in [19]. (The polynomial $p(A)$ will have degree $d-1$ and $Ap(A)$ will have degree d.) When iterating to solve $Ax = \tilde{b}$, GMRES picks an approximate solution $\tilde{x}$ from the Krylov subspace $\mathcal{K}_d(A, \tilde{b}) = span\{\tilde{b}, A\tilde{b}, A^2\tilde{b}, \ldots, A^{d-1}\tilde{b}\}$. Then there is a polynomial $p(A)$ such that $\tilde{x} = p(A)\tilde{b}$. Since GMRES minimizes the residual 2-norm $\|r\| = \|b - A\hat{x}\| = \|(I - Ap(A))b\|$, as $\|r\|$ goes to zero, we have that $Ap(A) \approx I$. This is why we choose $p(A)$ to be our preconditioner. We must generate the polynomial with a random starting vector $\tilde{b}$ rather than the problem right-hand side b. A structured starting vector b may not have components in the direction of every eigenvector of A and is prone to providing a polynomial that ignores parts of the spectrum of A. Unless the starting vector is random, using it to generate a polynomial typically proves ineffective [19].

Throughout this paper, we use right preconditioning to solve the following system:

$$\begin{aligned} Ap(A)y &= b, \\ x &= p(A)y. \end{aligned} \tag{2.1}$$

Because $p(A)$ is generated using GMRES, it can easily be composed with another preconditioner. When combined with a preconditioner M (such as ILU or block-Jacobi), the polynomial preconditioned system becomes

$$\begin{aligned} AMp(AM)y &= b, \\ x &= Mp(AM)y. \end{aligned} \tag{2.2}$$

There are two possible polynomial modifications for stability: added roots and damping. When the polynomial $Ap(A)$ has an extremely steep slope, it may become very ill-conditioned and sensitive to perturbations in floating-point arithmetic. This problem can be mitigated by adding extra copies of the roots (harmonic Ritz values) near the steep slope. Turning a single root into a larger multiplicity root flattens the polynomial $Ap(A)$ so that its application can be stable even for very high degrees. Root-adding can be fully automated via an approximation of the derivative of the polynomial at each harmonic Ritz value [5]. A different problem can occur where the operator A has a definite spectrum and $Ap(A)$ is indefinite, meaning the preconditioned problem is much more difficult for GMRES. In this case, the polynomial can be adjusted through damping; that is, the polynomial can be recomputed by running GMRES d steps on the right-hand side $A\tilde{b}$. This procedure frequently smooths high oscillations in the polynomial that create a more difficult spectrum.

The procedure to generate and apply the polynomial $p(A)$ of degree $d-1$ is detailed as follows: [19]

1. Run d steps of GMRES on the matrix A, using a random right-hand side $\tilde{b}$. (If damping is needed, use the right-hand side $A\tilde{b}$. To combine with another preconditioner M, run d steps of GMRES using the preconditioned operator AM.) This produces the Arnoldi decomposition $AV_d = V_{d+1}H_{d+1,d}$.

2. Compute the harmonic Ritz values θ_i of A (or AM) by solving for the eigenvalues of $H_{d,d} + h_{d+1,d}^2 f e_d^T$, where $f = H_d^{-*} e_d$ and $e_d = [0, \ldots, 0, 1]^T$.

3. Adding roots: For each θ_j, compute $prod(j) = \prod_{i=1, i \neq j}^{d} |1 - \theta_j/\theta_i|$. Let $k_j = \lceil (log_{10}(prod(j)) - 4)/14 \rceil$. If $k_j \geq 1$, insert k_j copies of root θ_j.

4. Order the θ_i's using a (Modified) Leja ordering [3], spacing repeated roots throughout.

5. Use the θ_i's to apply the preconditioner $p(A)$ via the formula

   (2.3)
$$p(A) = \sum_{k=1}^{d} \frac{1}{\theta_k} \left(I - \frac{1}{\theta_1} A\right) \left(I - \frac{1}{\theta_2} A\right) \cdots \cdots \left(I - \frac{1}{\theta_{k-2}} A\right) \left(I - \frac{1}{\theta_{k-1}} A\right).$$

   For real matrices with complex harmonic Ritz values θ_i, combine complex conjugates so that applying $p(A)$ requires only real arithmetic.

2.2 Implementation Trilinos [7] is open-source scientific computing software used in many large-scale applications including ice sheet modeling, circuit simulation, and fluid flows. Trilinos provides parallel iterative solvers, algebraic preconditioners, multigrid, eigensolvers, and many other capabilities. Linear algebra implementations, like Epetra and Tpetra, support distributed memory computations, while Kokkos Kernels supports shared memory (on-node) operations. Together, this allows for portable parallel linear algebra and graph operations, also on GPUs and other advanced architectures.

We implement the polynomial preconditioner in the Belos linear solvers package so that it is compatible with multiple linear algebra implementations and can be used in conjunction with other preconditioners. While the focus of this paper is on GMRES, the polynomial preconditioner can be used with all Krylov solvers available in Trilinos. Previously, only Chebyshev and least-squares polynomials were available in Trilinos as polynomial preconditioners. These polynomials only work with symmetric matrices and are only available when using Epetra.

Our implementation in Trilinos differs from that of [19] in that we always use formula 2.3 to apply $p(A)$ in the GMRES iteration. However, A and $p(A)$ are almost always applied consecutively in GMRES, so [19] uses a simpler formula for $Ap(A)$ to combine the operations at less expense. While our approach may require up to twice as many AXPYs, it is more intuitive and avoids stability issues that might arise from using two different formulas to compute $p(A)$. It is also compatible with the native stopping criteria checks provided by the Belos linear solvers.

For stability options, our implementation defaults to adding needed roots, but damping must be activated manually. Low degree ($d \leq 30$) polynomials typically need at most one or two added roots to be stable. The number of extra roots needed often increases as the degree of the polynomial increases. Occasionally the algorithm does create extra SpMV expense by adding a root unnecessarily, but the benefits of added roots on convergence typically outweigh the cost. See [5] for detailed examples. Damping, on the other hand, is usually unnecessary; it is rare for a GMRES polynomial to make a definite problem indefinite. Damped polynomials have less of the erratic behavior that comes with high oscillations, but they do not separate small eigenvalues from the origin as well as un-damped polynomials. This means that damping may not give the best spectrum for GMRES convergence if it is not needed to fix a problem. Thus, damping must come with a precaution: when an un-damped polynomial works well, its damped counterpart will need significantly more iterations for GMRES to converge. Often the need for damping is displayed when some polynomial degrees help GMRES converge but other degrees stall the solver. Indefinite matrices may also benefit from damping if no un-damped polynomials are successful.

3 Robustness and Cost Reduction with Polynomial Preconditioning

For all examples, we run GMRES with two steps of classical Gram-Schmidt orthogonalization (ICGS(2)). This requires two block dot products and one norm per iteration. We use the Zoltan package for hypergraph partitioning and load-balancing. This helps optimize the communication pattern for the matrix-vector product. Times for load-balancing, partitioning, and preconditioner creation are not included in total solve times. These costs would be amortized over a full-scale simulation with multiple linear solves. In our discussion, "polynomial degree" indicates the degree d of $Ap(A)$. Automatic root-adding for stability is enabled throughout the following experiments. Typically, the number of added roots is less than 4, so we do not explicitly make note of added roots when indicating the polynomial degree. The additional matrix-vector products due to added roots are included in final counts of SpMVs. Experiments in this section were run on Baylor University's Kodiak cluster. The compute nodes have dual 18-core Intel E5-2695 V4 Broadwell processors with 256 GB of RAM. All tests in this section were run using 32

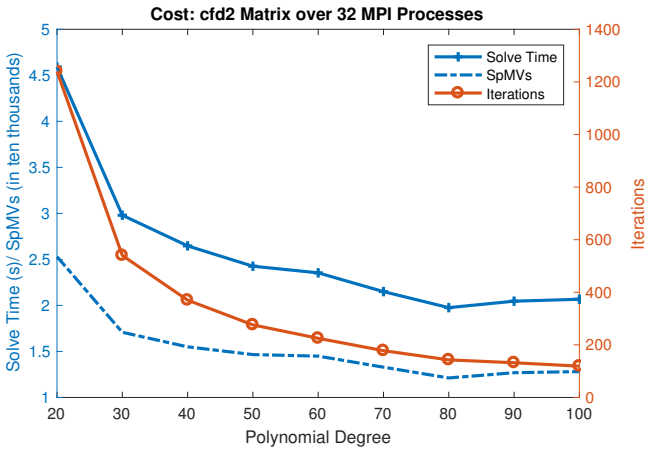

Figure 1: Solve time (crosses), SpMVs (dotted), and Iteration count (circles) for matrix cfd2 with varying degrees of polynomial preconditioner. The left vertical axis corresponds to Solve Time (in seconds) and SpMVs (in ten thousands). The right vertical axis corresponds to iterations.

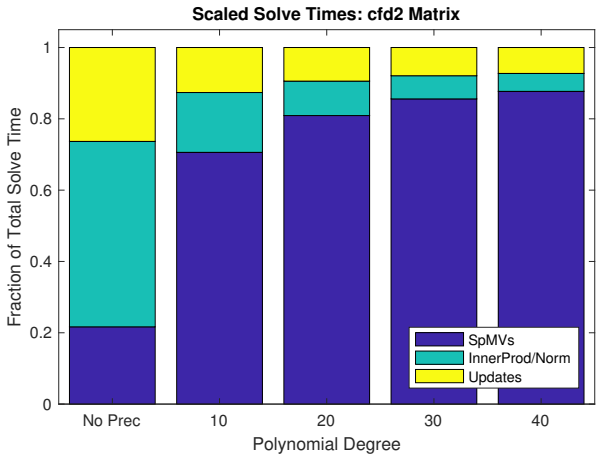

Figure 2: Distribution of total solve time for matrix cfd2 using no preconditioning and polynomials of increasing degree. The 1 on the vertical axis represents total solve time. The bottom (purple) section of each bar represents fraction of total solve time spent in SpMVs, the middle (teal) section time spent in inner products and norms, and the top (yellow) section time spent for vector updates (AXPYs).

MPI processes on a single compute node.

3.1 A CFD example We first demonstrate the potential cost savings from polynomial preconditioning with a small computational fluid dynamics example. The matrix cfd2 from the Rothberg group in the SuiteSparse Matrix Collection [4] is symmetric positive definite and has dimension $n = 123{,}440$. We run GMRES(50) with a random right-hand side b to a tolerance of 1e−8. Without preconditioning, GMRES needs 113.8 seconds and 171,541 iterations to converge. Even a degree 10 polynomial preconditioner gives almost ten times improvement with a solve time of 11.43 seconds, using only 5436 iterations and 55,460 SpMVs. The degree 20 polynomial gives another two times speedup with a solve time of 4.6 seconds and 1240 iterations.

As illustrated in Figure 1, the solve time, SpMVs and iteration count continue to drop with increasing polynomial degrees. The drop is sharp from degree 20 to $d = 30$ and then slower as the degree continues to increase. The best solve time occurs with $d = 80$ at 1.98 seconds with 12,120 SpMVs and 143 iterations. Recall that the iteration count corresponds to the communication-intensive orthogonalization cost of norms and dot products. The 143 iterations for a degree 80 polynomial represent more than thirty-eight times reduction of dot products and norms over the degree 10 polynomial.

After degree 80, we reach a point of diminishing returns; while iterations and orthogonalization continue to decrease, this does not give improved performance. Beginning with degree 90, SpMVs begin to increase slightly, and so does solve time. For example, the degree 100 polynomial converges in only 119 iterations, but its 12,810 SpMVs cause it to converge in 2.07 seconds. This behavior is typical of examples over a fixed number of processors: high degree polynomials often continue to improve the iteration count even when they are no longer reducing SpMVs and solve time. When applied in a large-scale computing situation, the trade-offs for solve time become less clear. The additional SpMVs of a high-degree polynomial could be justified for the sake of avoiding global communication.

Figure 2 shows the proportion of time spent in three different kernels: SpMVs, inner products/norms, and vector updates (AXPYs). Notice that with no preconditioning, 52% of the solve time is spent performing communication-intensive inner products and norms. With a polynomial of degree 10, about 71% of solve time is spent in SpMVs with only 17% of solve time spent in inner products and norms. As the degree increases, this balance continues to shift further towards SpMVs, which take 88% of solve time with degree 40. Since this experiment was run on one compute node, actual communication costs are minimized. However, this distribution foreshadows the improvement that polynomial preconditioning can bring at large-scale when global reductions and synchronizations become far more costly. Additionally, the local communication costs of the Sp-

MVs can be further reduced by using a Matrix Powers Kernel, which will be discussed in section 5.

For comparison to other preconditioners, solve time for cfd2 with a block Jacobi preconditioner is 8.19 seconds with 951 iterations. This is faster than the degree 10 polynomial, but slower than degree 20. The iteration count is less than for degree 20, but higher than with degree 30. Combining the Jacobi preconditioner with a degree 5 polynomial reduces solve time to 2.78 seconds and 68 iterations, about 2.5 seconds of which consists of applying the Jacobi preconditioner. This solve time is higher than with the degree 40 polynomial, but the iteration count is lower than that of even the degree 100 polynomial. Higher degrees of polynomial combined with this preconditioner continued to reduce the iteration count, but did not provide any further decrease in solve time. Although block Jacobi alone is effective for this problem, polynomial preconditioning makes it even more worthwhile.

3.2 Comparing to Chebyshev Polynomials Using the same cfd2 matrix, we compare the GMRES polynomial preconditioner with a Chebyshev polynomial from the Ifpack package. (GMRES polynomials in this subsection were generated with a different starting vector than above, so timings and convergence differ.) We used the Ifpack Chebyshev preconditioner as a "black box." Internally, it diagonally scales the problem matrix and runs 10 iterations of the power method to estimate the largest eigenvalue λ_{max}.

The Chebyshev polynomials, regardless of degree, were cheaper to generate: Chebyshev polynomial creation always required less than 0.01 seconds, while GMRES polynomials needed from 0.012 seconds (degree 5) to 0.09 seconds (degree 60). More important is the difference in solve times, plotted in Figure 3. For degrees 5 and 10, the Chebyshev polynomials win by a wide margin, using roughly half the iterations and solve time of the GMRES polynomials. However, GMRES polynomials give the best timings overall. For degree 30 and above, the GMRES polynomials win, solving in under 3 seconds while Chebyshev polynomials need over 7 seconds. The Chebyshev polynomial of degree 60 needs 696 iterations while the GMRES polynomial of degree 60 needs only 225 iterations.

It is interesting that a Chebyshev polynomial works best at low degrees and the GMRES polynomial is better at higher degrees. We observed similar patterns with a 3D Laplacian. The diagonal scaling with the Chebyshev polynomial may have skewed the results. Further investigation is needed to study which polynomial creates the best preconditioned spectrum for GMRES and whether this determines the best timings. It is also possible that the Chebyshev polynomial creates a good spectrum but is simply less stable for high degrees.

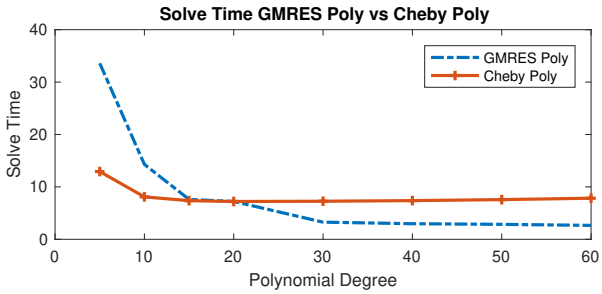

Figure 3: Solve times comparing GMRES polynomial and Chebyshev polynomial with different degrees for matrix cfd2.

Matrix	n:	NNZ:	Symm:
cfd2	123,440	3,085,406	SPD
Transport	1,602,111	23,487,281	none
FEM_3D_thermal2	147,900	3,489,300	none
thermal2	1,228,045	8,580,313	SPD
xenon2	157,464	3,866,688	none
ML_Geer	1,504,002	110,686,677	none
G3_circuit	1,585,478	7,660,826	SPD

Table 1: Size and format of matrices tested from SuiteSparse.

3.3 A Large Test Set We now demonstrate polynomial preconditioning with a set of seven matrices from the SuiteSparse Matrix Collection [4]. Information about the size and structure of each matrix is in Table 1. If provided, we used the given right-hand side vector from SuiteSparse. Otherwise, we generated a random right-hand side vector. We run GMRES(100) to a tolerance of 1e−8.

Various preconditioning scenarios were considered in this study. We constructed polynomials of degrees $20, 40$, and 60 with no additional preconditioning, and we also tested an ILU(k) preconditioner from Ifpack by itself and composed with polynomials of degrees $5, 10$, and 20. Results for the polynomial with no preconditioning are in Table 2. For matrices ML_Geer, thermal2, and G3_Circuit, GMRES(100) with no preconditioning did not converge within 100,000 iterations, so "xxx" indicates the failure of the solver.

Results for the study using the ILU(k) preconditioner are presented in Table 3. The ILU preconditioner had fill level 1 and used Additive Schwarz decomposition with an overlap of 1. For matrices cfd2, xenon2, and ML_Geer, the ILU preconditioned problem either

Matrix	No Prec		Degree 20			Degree 40			Degree 60		
	Time	Iters	SPMVs	Time	Iters	SPMVs	Time	Iters	SPMVs	Time	Iters
cfd2	102	85116	15740	3.82	779	11850	2.52	286	12600	2.533	198
Transport	1042	40268	6048	26.16	285	3948	15.19	93	4032	14.51	63
FEM_3D_thermal2	0.73	480	630	0.151	29	748	0.16	16	804	0.193	11
thermal2	xxx	xxx	26200	78.98	1297	14280	37.01	353	6466	17.1	104
xenon2	5.73	3304	2080	0.626	102	2200	0.521	54	2340	0.606	38
ML_Geer	xxx	xxx	260500	3214	12897	61580	731.5	1487	29570	346.7	472
G3_circuit	xxx	xxx	xxx	xxx	xxx	20080	35.88	473	xxx	xxx	xxx
G3_circuit DAMPED	N/A	N/A	47380	108.7	2233	32100	57.74	775	28730	45.82	451

Table 2: Solve times and statistics for several SuiteSparse matrices and polynomial degrees.

did not converge or converged more slowly than with no preconditioning. We now highlight key insights from each of the matrix experiments:

cfd2: This is the same matrix from example 3.1, but GMRES uses a subspace size of 100 instead of 50. Across the board, solve times are faster with the large subspace, using less iterations and SpMVs. However, with polynomial preconditioning, the point of diminishing returns comes sooner. SpMVs and solve time go up starting with degree 60 instead of degree 90. This suggests that high-degree polynomials may be more effective with smaller subspaces where more power is required to overcome the information lost with restarting.

Transport: For this problem, solve time decreases from degree 40 to degree 60 even though the number of SpMVs has slightly increased. ILU(1) preconditioning alone is less effective than the polynomial preconditioners alone, but the best solve time occurs when combining polynomials with ILU(1). You can see the breakdown of relative solve times with ILU plus PP-GMRES in Figure 4. With ILU(1) and no polynomial, applying ILU takes 21% of solve time, while inner products and norms need 31% of solve time. When composed with a degree 20 polynomial, ILU dominates the solve time at 63% while inner products and norms are reduced to 3% of solve time. Multiplications with A consume less than 30% of solve time, even when we increase the polynomial degree to 40. Thus, combining PP-GMRES with an expensive preconditioner, like ILU, places most of the work in the ILU kernel.

Polynomial preconditioner creation times seem reasonable compared to ILU. For this matrix, the ILU preconditioner alone required about 0.32 seconds to create. To compose ILU with a degree 5 polynomial requires an additional 0.06 seconds, and to compose it with a degree 40 polynomial requires an additional 0.73 seconds. By comparison, constructing a polynomial of degree 20 without ILU requires 0.18 seconds, degree 40 needs 0.49 seconds, and degree 80 uses 1.66 seconds.

FEM_3D_thermal2: For this matrix, polynomial preconditioning alone works well, but ILU(1) is significantly better. The ILU preconditioner is so effective that adding a polynomial does not give noticeable speedup. For polynomials without ILU, this matrix used the most added roots for stability. It needed 1 added root for degree 20, 4 roots with degree 40, and 7 added roots for degree 60. No other matrices in the test set required a degree 60 polynomial with more than 3 added roots or a degree 40 polynomial with more than 2 added roots.

thermal2: In this instance, ILU(1) applied alone is the clear loser. It needs almost ten times as many SpMVs and over three times the solve time of the degree 20 polynomial. However, high-degree polynomials applied alone give comparable results to ILU(1) plus low-degree polynomials. In terms of solve time and iterations, the degree 60 polynomial is almost equivalent to ILU(1) plus degree 20. To balance the trade-offs at large scale, one would need to consider whether it is cheaper to do more SpMVs with the matrix A alone versus fewer SpMVs with AM, where M is the ILU preconditioner.

xenon2 and ML_Geer: For both of these matrices, polynomial preconditioning gives good improvement, while ILU(1) makes convergence worse. The ML_Geer problem is especially difficult. While low-degree polynomials work well, degree 80 gives good improvement over degree 60, with 16,970 SpMVs, 200 iterations, and 197 seconds solve time. For this matrix, the point of diminishing returns occurs after degree 120, which gives a solve time of 141.5 seconds. This suggests that very high-degree polynomials can be useful for challenging problems. Notice also the improvement in iterations and dot products: Doubling the polynomial degree from 20 to 40 gives over 8.6 times reduction in dot products, and doubling from degree 40 to degree

Matrix	ILU Only		ILU + Deg 5			ILU + Deg 10			ILU + Deg 20		
	Time	Iters	SPMVs	Time	Iters	SPMVs	Time	Iters	SPMVs	Time	Iters
Transport	55.9	1879	1595	22.59	315	920	10.81	91	960	10.22	47
FEM_3D_thermal2	0.06	20	30	0.099	5	40	0.058	3	45	0.064	2
thermal2	301	12902	7310	64.73	1447	4710	34.58	466	2180	16.52	107
G3_circuit	23.9	982	1120	9.474	221	1160	7.502	114	1080	5.484	53

Table 3: Solve times and statistics for ILU and polynomial preconditioning.

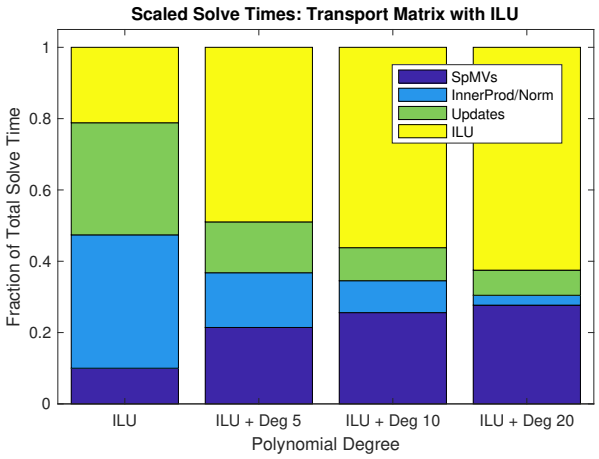

Figure 4: Relative solve times (scaled) for matrix Transport for ILU and ILU combined with polynomials of varying degrees.

80 gives another 7.4 times reduction.

G3_Circuit: Of particular interest is the matrix G3_Circuit. With stand-alone polynomial preconditioning, degree 40 gives convergence while degrees 20 and 60 do not. With damping, (see last row in Table 2), polynomials of degrees 20, 40, and 60 all give convergence, but degree 40 converges more slowly than without damping. This is expected because a damped polynomial usually gives a more difficult spectrum than an effective un-damped polynomial. Ultimately, the best solution for this problem was to combine with ILU(1) preconditioning. Solving the ILU preconditioned problem proves to be faster than with any of the polynomials working alone. Composing it with a degree 20 polynomial further improves the solve time by four times and the iteration/orthogonalization count by 18.5 times.

3.4 Combining with a Simple Multigrid

Factorization-based preconditioners are not the only preconditioners that can be composed with polynomials. As long as algebraic multigrid (AMG) is applied consistently, it, too, can be combined with a polynomial. (Polynomials should not be combined with any preconditioner that requires FGMRES; to do so would require re-creating the polynomial each time the inner preconditioner changes.) We use the MueLu package of Trilinos to combine algebraic multigrid with a polynomial preconditioner. The problem is a 3D Laplacian generated with the Galeri package. It has a unit cube mesh with 250 grid points in each direction and size $n = 15{,}625{,}000$. The right-hand side is randomly generated. The algebraic multigrid uses smoothed aggregation with Chebyshev smoothing over 5 levels on 32 MPI processes. We run GMRES(50) to a tolerance of 1e−8.

With multigrid preconditioning, the problem converges in 42 iterations and 13.95 seconds. Setup for the multigrid preconditioner takes about 0.95 seconds. Composing with even a degree 2 polynomial gives noticeable improvement. Table 4 gives timings and iteration counts for several degrees. The polynomial of degree 10 gives the best solve time of 7.62 seconds with 6 iterations. The point of diminishing returns comes quickly compared to previous examples; degrees 12 and above give slightly slower convergence than degree 10. These results are not surprising since multigrid is more costly to apply than ILU. With the degree 10 polynomial, applying multigrid uses about 73% of the solve time. Using higher degree polynomials means spending more time in multigrid with fewer basis vectors saved. Polynomial preconditioners combined with multigrid are more expensive to create than those combined with ILU. When we add the polynomial creation and solve times, the degree 5 polynomial with AMG is fastest at 8.53 seconds. While this problem shows only minimal benefit from polynomial preconditioning, it foreshadows the potential improvement for problems such as convection-diffusion that are more difficult for AMG.

Aggregating results of the previous examples suggests a strategy for choosing polynomial degrees: When polynomial preconditioning is deployed alone, degrees of 40 and higher are often appropriate. When combined with a simple factorization-based preconditioner such as ILU or block Jacobi, degrees from 5 to 30 seem to be most effective. Combining a polynomial with multigrid

	Iters	Solve Time	Poly Create	Solve + Poly Create
AMG only	42	13.95		13.95
AMG + Deg 2	26	9.71	0.29	10.00
AMG + Deg 3	19	8.55	0.44	8.99
AMG + Deg 5	12	7.78	0.75	8.53
AMG + Deg 7	9	7.90	1.10	9.00
AMG + Deg 10	6	7.62	1.69	9.31
AMG + Deg 12	4	7.75	2.14	9.89

Table 4: Solve time, iterations, and polynomial creation time for AMG with polynomial preconditioning.

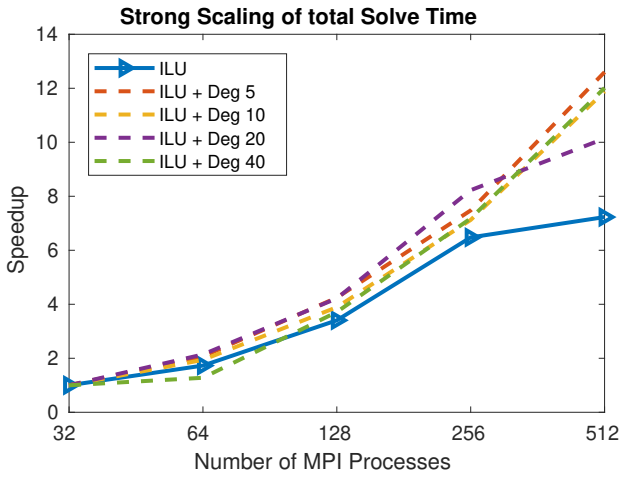

Figure 5: Strong scaling for total solve time of a 2D convection-diffusion problem, computed as solve time over N cores divided by solve time over 32 cores. Scaling is shown for ILU(1) preconditioning alone versus ILU combined with various polynomials.

makes each polynomial application very expensive, so small degrees from 2 to 15 are likely to be best.

4 Larger-Scale Experiments

In this section, experiments were run on a cluster where the compute nodes each have 2.6 GHz Intel E5-2670 Sandy Bridge processors with 16 cores and 64 GB of RAM. These experiments were run with GMRES(50) to a tolerance of 1e−8. Problem matrices were generated with the finite difference stencils in the Galeri package. All ILU preconditioners have a fill level of 1 and Additive Schwarz overlap of 1.

We read in a randomly generated vector from file to use as the right-hand side for both problems. We also use this vector as the starting vector to generate the polynomial. This prevents inconsistencies that may arise when random number generators create different vectors as the number of MPI processes increases. Note that even with the same generating vector, a degree d polynomial may have minor differences over different numbers of MPI processes. In exact arithmetic, two degree d polynomials generated from the same starting vector will be the same. This is unlike an ILU preconditioner with domain decomposition, which is guaranteed to change with increasing MPI processes and subdomains. However, varying dot-product reduction trees over more processes can cause small differences in the polynomial, just as they can cause unpreconditioned GMRES to have somewhat different convergence when varying the number of processors.

4.1 Convection-Diffusion: Our first example is the convection-diffusion "UniFlow2D" problem. We use a square mesh with 1000 grid points, resulting in a matrix of size $n = 100$ million with 449,960,000 nonzeros. While polynomial preconditioning alone did help to improve convergence, the iteration count and solve times were high. For instance, over 128 cores, a degree 40 polynomial needed 728 iterations and 943 seconds to converge. Polynomials worked much better when combined with an ILU(1) preconditioner.

We tested ILU(1) alone, and then combined it with polynomials of degree $5, 10, 20, 40$ and 80. Tests used from 2 to 32 nodes, each with 16 cores, for a total of 32 to 512 MPI processes. Adding a degree 5 polynomial gave anywhere from 2.8 to 4.9 times improvement in solve time over ILU alone. Over 2 nodes, ILU(1) alone converges in 252.9 seconds (208 iterations), and it uses 35 seconds (327 iterations) over 32 nodes. ILU(1) plus a degree 10 polynomial gives the fastest convergence with 64.24 seconds (17 iterations) over 2 nodes and 5.41 seconds (24 iterations) over 32 nodes. ILU with polynomials of degrees $20, 40$, and 80 does run slower. This supports the assertion that from section 3.4 that ILU is best combined with not-too-large of a polynomial degree. It is notable, however, that the ILU with a degree 80 polynomial converges in only 3 to 5 iterations.

ILU with polynomial preconditioning does show better strong scaling than ILU alone (Figure 5). This is likely because the polynomials help to maintain more consistent iteration counts over additional cores and mitigate the effects of domain decomposition on the ILU preconditioner. We do not believe that global communication played a significant role in these scaling results. However, it is notable that polynomial preconditioning may help to overcome the scaling difficulties of ILU.

4.2 3D Laplacian: We also test a standard finite difference 3D Laplacian from the Galeri package. We

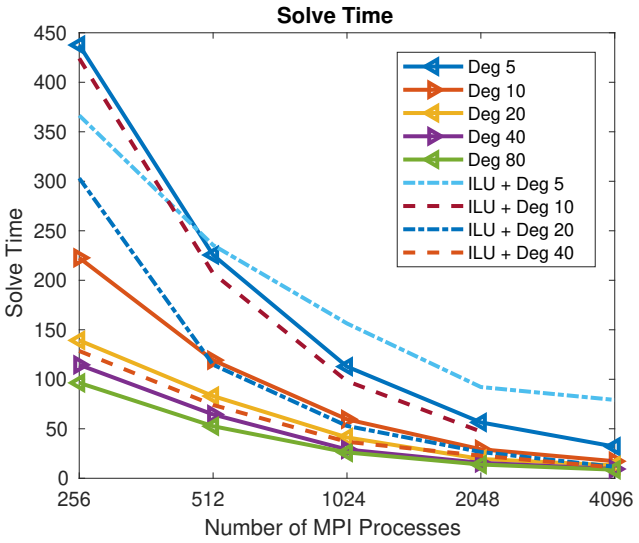

Figure 6: Total solve time for polynomial preconditioners of varying degrees (some with ILU) applied to a 3D Laplacian over increasing numbers of MPI processes.

use a cube mesh with 550 grid points. This gives a matrix of size $n = 166{,}375{,}000$ with over 1.1 billion nonzeros. This problem was run on 16 nodes up to 256 nodes (256 to 4096 cores). All polynomial preconditioners were more effective than ILU(1) with no polynomial. Combining the polynomials with ILU gave improvement, but solve times were interspersed with those of polynomials working alone, similar to the thermal2 problem. Several solve times are shown in Figure 6. We observe that high-degree polynomials seem to make less of an impact to solve time at high core counts than at low core counts. The degree 80 polynomial clearly converges fastest over 256 cores, but beginning with 2048 cores, it is slightly overtaken by the degree 60 polynomial [not plotted] and is very close in run time to the degree 40 polynomial. Even so, the degree 40 polynomial still represents a three times improvement in solve time over the degree 5 polynomial. Further study is required at larger scale to analyze the trade-offs between a more expensive matrix-vector product versus reducing dot products and global synchronizations.

Strong scaling for the polynomial preconditioned problems was near perfect and roughly equivalent to the strong scaling of non-preconditioned GMRES. This indicates that global communication is not a bottleneck at this scale on this computing cluster. Strong scaling for ILU with polynomials was mixed. ILU combined with degree 5 did not maintain consistent iteration counts as well as ILU combined with higher degree polynomials. In fact, over 256 nodes, ILU alone converges slightly faster than ILU plus a degree 5 polynomial. ILU plus a degree 10 polynomial performs well for up to 2048 cores, but then doesn't converge for 4096 cores. Further investigation is needed to understand this failure. The ILU preconditioner alone scaled worse than the problems with only polynomial preconditioning.

5 Relation to other Communication-Avoiding Methods

There are several variants of GMRES which avoid communication inherent in the standard implementation. These methods reduce the number of global communication steps at the cost of more memory and flops. The savings in global reduction and synchronization is important on extreme-scale parallel systems. In this section, we discuss polynomial preconditioning in comparison to and in combination with three other communication-reducing strategies. Two are GMRES variants, pipelined GMRES and s-step GMRES, that have similarities to PP-GMRES that can be exploited. The third strategy is the Matrix Powers Kernel (MPK) [8] which can minimize local communication in the many SpMVs required to apply the polynomial.

As discussed before, PP-GMRES itself avoids communication. The polynomial preconditioner mitigates all-to-all communication and synchronization from dot products and norms by performing more work between each orthogonalization step. However, unlike other communication-avoiding GMRES variants, PP-GMRES is not mathematically equivalent to standard GMRES, nor does it give the same convergence in terms of SpMVs. For instance, we do not expect GMRES(50) with a degree 6 polynomial to converge in as few SpMVs as GMRES(300). In this case, PP-GMRES minimizes the problem residual over only a subset of the subspace that GMRES(300) uses. Nevertheless, polynomial preconditioning can be very effective in reducing orthogonalization and overcoming the limitations imposed by restarting GMRES.

Pipelined Methods: Pipelined methods hide global communication by overlapping it with other calculations, typically to include SpMVs. A one-level pipelined GMRES method in [6] combines the block dot product and norm for one orthogonalization step and overlaps this with the SpMV for creating the next Krylov basis vector. Sometimes dot product latency is longer than the time to compute a single SpMV, leaving processors idle as the synchronization completes. To compensate for this, one can extend the "pipeline length" by combining the dot products of ℓ iterations into one step, overlapping this communication step with ℓ SpMVs. This is the idea behind the $p(\ell)$-GMRES algorithm in [6]. While this strategy is advantageous in

terms of keeping processors running, longer pipelines require more calculations, more vector storage, and introduce more sources of instability. In addition to the extra computations required to use overlapping dot products, one must compute shifts to maintain linear independence of basis vectors and avoid breakdown of the method.

Adding preconditioning to pipelined GMRES can be advantageous because the preconditioned matrix-vector product will be more expensive than a single SpMV. This means that shorter pipeline lengths will be needed to hide communication latency and the method will have better stability. Polynomial preconditioning may be better suited than other preconditioners to pipelined GMRES for two reasons: 1) Some preconditioners can require global communication to apply. This defeats the intention of pipelined methods; one cannot hide global synchronizations with even more global communication. Since polynomial preconditioning requires only local communications, it can be easily overlapped with the dot products. 2) Polynomial degree selection may give flexibility to obtain a preconditioner tuned to the necessary pipeline length. One could select a polynomial degree based upon the number of SpMVs required to hide the latency of one dot product.

s-step Methods: s-step GMRES (see [8] and citations therein) takes a different approach to avoiding communication. Like pipelined GMRES, it consecutively performs several SpMVs for the next s Krylov basis vectors. But instead of trying to overlap orthogonalization with SpMVs, all the new basis vectors are orthogonalized at once using the Tall-Skinny QR (TSQR) algorithm. Thus global communication happens only once every s iterations. Like the pipelined methods, s-step GMRES is also prone to numerical instability as the step size s increases. The vector sequence $b, Ab, A^2b, .., A^sb$ will quickly lose linear independence without orthognalization. To maintain stability and linear independence, extra computations are performed to create a Newton Basis [3].

Polynomial preconditioning is similar to s-step GMRES in that orthogonalization is only performed once every d SpMVs after applying the polynomial (of degree d). However, s-step GMRES is mathematically equivalent to standard GMRES, while polynomial preconditioned GMRES is not. Polynomial preconditioning could be used within s-step GMRES. In this case, orthogonalization is only needed once every $d*s$ SpMVs. This PP-s-step GMRES may give the best balance between minimizing SpMVs, delaying orthogonalization, and maintaining stability.

Matrix Powers Kernel and CA-GMRES: The Matrix Powers Kernel (MPK) is designed to perform several matrix-vector products consecutively while minimizing reads from memory. With the MPK, local communication can be reduced at a cost in increased storage requirement. Communication-Avoiding (CA)-GMRES [8] is a GMRES variant which combines the MPK with s-step GMRES. Another possibility is to use the MPK with polynomial preconditioning to evaluate $Ap(A)$. This could be combined with either s-step or pipelined GMRES. When the MPK is used to evaluate a polynomial, rather than to take s steps as in CA-GMRES, cache reuse is improved because intermediate vectors do not need to be stored. Thus, PP-GMRES provides an avenue for taking advantage of the MPK while avoiding the numerical pitfalls of delayed orthogonalization that occur in s-step and pipelined methods.

6 Conclusions and Future Work

We described an implementation of the GMRES polynomial preconditioner in Trilinos and showed parallel results. We demonstrated that polynomial preconditioning is effective and stable for a variety of practical problems and that it reduces global communication from dot products and norms. Our experiments show that the polynomials are versatile and adaptable with many preconditioning combinations. We also discussed potential to enhance current communication-avoiding Krylov solvers.

Future work includes testing on very large-scale applications and exploration with GPUs (where polynomial methods should be well suited). In particular, we would like to analyze the effectiveness of polynomial preconditioning combined with s-step GMRES in avoiding global communication. Other potential expansions include: a) modifying the polynomial to precondition large eigenvalue problems in the Anasazi package; b) automating the "damping" parameter; and c) testing the GMRES polynomial as a smoother for algebraic multigrid. We expect that the GMRES polynomial preconditioner will become a useful addition to the Trilinos library.

7 Acknowledgments

Thanks to Ichitaro Yamazaki and Christian Glusa for help constructing experiments with multigrid and with s-step GMRES. Thanks to Ron Morgan and Rob Kirby for feedback and suggestions.

References

[1] A. M. Abdel-Rehim, R. B. Morgan, and W. Wilcox, *Improved seed methods for symmetric positive definite linear equations with multiple right-*

hand sides, Numer. Linear Algebra Appl., 21 (2014), pp. 453–471.

[2] S. F. Ashby, *Polynomial preconditioning for conjugate gradient methods*, ProQuest LLC, Ann Arbor, MI, 1988. Thesis (Ph.D.)–University of Illinois at Urbana-Champaign.

[3] Z. Bai, D. Hu, and L. Reichel, *A Newton basis GMRES implementation*, IMA J. Numer. Anal., 14 (1994), pp. 563–581.

[4] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), pp. Art. 1, 25.

[5] M. Embree, J. A. Loe, and R. B. Morgan, *Polynomial preconditioned Arnoldi*. Preprint, `https://arxiv.org/abs/1806.08020`.

[6] P. Ghysels, T. J. Ashby, K. Meerbergen, and W. Vanroose, *Hiding global communication latency in the GMRES algorithm on massively parallel machines*, SIAM J. Sci. Comput., 35 (2013), pp. C48–C71.

[7] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.

[8] M. F. Hoemmen, *Communication-avoiding Krylov subspace methods*, Tech. Rep. UCB/EECS-2010-37, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, April 2010.

[9] O. G. Johnson, C. A. Micchelli, and G. Paul, *Polynomial preconditioners for conjugate gradient calculations*, SIAM J. Numer. Anal., 20 (1983), pp. 362–376.

[10] W. Joubert, *A robust GMRES-based adaptive polynomial preconditioning algorithm for nonsymmetric linear systems*, SIAM J. Sci. Comput., 15 (1994), pp. 427–439. Iterative methods in numerical linear algebra (Copper Mountain Resort, CO, 1992).

[11] A. A. Kamiabad and J. E. Tate, *Polynomial Preconditioning of Power System Matrices with Graphics Processing Units*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 229–246.

[12] R. Li and Y. Saad, *GPU-accelerated preconditioned iterative linear solvers*, The Journal of Supercomputing, 63 (2013), pp. 443–466.

[13] R. Li, Y. Xi, E. Vecharynski, C. Yang, and Y. Saad, *A thick-restart Lanczos algorithm with polynomial filtering for Hermitian eigenvalue problems*, SIAM J. Sci. Comput., 38 (2016), pp. A2512–A2534.

[14] Y. Liang, *Stability of polynomial preconditioning*, in Proceedings of Algoritmy 2000, Conference on Scientific Computing, A. Handlovicova, M. Komornikova, K. Mikula, and D. Sevcovic, eds., Comenius Univ. Press, 2001, pp. 264–273.

[15] ———, *The use of parallel polynomial preconditioners in the solution of systems of linear equations*, PhD thesis, University of Ulster, 2005.

[16] Y. Liang, M. Szularz, and L. T. Yang, *Finite-element-wise domain decomposition iterative solvers with polynomial preconditioning*, Math. Comput. Modelling, 58 (2013), pp. 421–437.

[17] Y. Liang, J. Weston, and M. Szularz, *Generalized least-squares polynomial preconditioners for symmetric indefinite linear equations*, vol. 28, 2002, pp. 323–341. Parallel matrix algorithms and applications (Neuchâtel, 2000).

[18] Q. Liu, R. B. Morgan, and W. Wilcox, *Polynomial preconditioned GMRES and GMRES-DR*, SIAM J. Sci. Comput., 37 (2015), pp. S407–S428.

[19] J. A. Loe and R. B. Morgan, *New polynomial preconditioned GMRES*. Preprint, `https://arxiv.org/abs/1911.07065`.

[20] D. P. O'Leary, *Yet another polynomial preconditioner for the conjugate gradient algorithm*, Linear Algebra Appl., 154/156 (1991), pp. 377–388.

[21] H. Rutishauser, *Theory of gradient methods*, in Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems, M. Engeli, T. Ginsburg, H. Rutishauser, and E. Stiefel, eds., Birkhäuser, Basel-Stuttgart, 1959, pp. 24–49.

[22] Y. Saad, *Practical use of polynomial preconditionings for the conjugate gradient method*, SIAM Journal on Scientific and Statistical Computing, 6 (1985), pp. 865–881.

[23] ———, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd ed., 2003.

[24] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[25] H. K. Thornquist, *Fixed-polynomial approximate spectral transformations for preconditioning the eigenvalue problem*. PhD Thesis, Rice University, TR06-05, 2006.

[26] M. B. van Gijzen, *A polynomial preconditioner for the GMRES algorithm*, J. Comput. Appl. Math., 59 (1995), pp. 91–107.

[27] J. Zhang and L. Zhang, *Efficient CUDA polynomial preconditioned conjugate gradient solver for finite element computation of elasticity problems*, Math. Probl. Eng., (2013), pp. Art. ID 398438, 12.