



Parallel Newton-Chebyshev Polynomial Preconditioners

Ángeles Martínez Calomardo

Department of Mathematics and Earth Sciences, University of Trieste

joint work with

Luca Bergamaschi

Department of Civil Environmental and Arch. Engineering, University of Padua

8th ECM Congress

21 June, 2021

1. Newton's method for matrix inversion.
2. From Newton's method to a polynomial preconditioner for the CG solver.
3. Relation with the Chebyshev polynomial preconditioner.
4. Minimizing the condition number is not enough. How to avoid clustering of smallest eigenvalues.
5. Serial Numerical Results.
6. Parallel Numerical Results.
7. Conclusions.

We deal with the solution of linear systems

$$Ax = b,$$

where A is very large, sparse and symmetric positive definite (SPD).

We use an iterative method: the Preconditioned Conjugate Gradient Method.

We look for a preconditioner in order to speed-up the iteration process with the following features in view of a (massively) parallel implementation:

- It must be **scalable** on modern HPC architectures (when increasing the number of processors we expect a consequent reduction on the overall CPU time).
- **No extra memory required** in addition to the PCG vectors.
- It must be **matrix-free**. Storing the coefficient matrix is not needed, only its application to a vector is required.

PDE constrained optimization problem modelling fluid flow in fractured porous media, once discretized with FE, gives rise to the algebraic linear system $\mathbf{K}\mathbf{x} = \mathbf{f}$, where

$$\mathbf{K} = \left[\begin{array}{cc|c} A & 0 & -C \\ G^h & A & -B \\ \hline -B^T & -C^T & G^u \end{array} \right], \quad \mathbf{x} = \begin{pmatrix} \mathbf{h} \\ \mathbf{p} \\ \mathbf{u} \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} \mathbf{q} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \quad (1)$$

G^h and G^u are SPSD, usually rank-deficient; B, C are rectangular coupling blocks A is SPD with a block diagonal structure. The system is rewritten as

$$\begin{bmatrix} M & -Z \\ -W^T & G^u \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{0} \end{bmatrix}, \quad \text{with } M = \begin{bmatrix} A & 0 \\ G^h & A \end{bmatrix}, Z = \begin{bmatrix} C \\ B \end{bmatrix}, W = \begin{bmatrix} B \\ C \end{bmatrix}^T, \mathbf{x}_1 = \begin{bmatrix} \mathbf{h} \\ \mathbf{0} \end{bmatrix}, \mathbf{f}_1 = \begin{bmatrix} \mathbf{q} \\ \mathbf{0} \end{bmatrix}.$$

$$\text{Block GE} \quad \rightarrow \quad \begin{bmatrix} M & -Z \\ 0 & \underbrace{G^u - W^T M^{-1} Z}_{S_u} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \underbrace{W^T M^{-1} \mathbf{f}_1}_{\mathbf{r}} \end{bmatrix}$$

This block triangular system requires solution of $S_u \mathbf{u} = \mathbf{r}$.

The SPD Schur complement matrix S_u must not be explicitly formed \rightarrow it calls for a matrix-free preconditioner

Small size example

	n	nnz	nnz per row
\mathbf{K}	754806	1.08×10^7	14
S_u	312518	3.25×10^8	1041

Newton's method for matrix inversion

The Newton-Raphson method for the scalar equation

$$x^{-1} - a = 0, \quad a \neq 0,$$

is

$$x_{j+1} = 2x_j - ax_j^2, \quad j = 0, \dots, \quad x_0 \text{ fixed.}$$

Newton-Schulz method for matrix inversion (1933) or Hotelling's method (1943)

Newton's method applied to $P^{-1} - A = 0$

$$P_{j+1} = 2P_j - P_jAP_j, \quad j = 0, \dots, \quad P_0 \text{ fixed,} \quad (2)$$

Starting from P_0 satisfying $P_0A = AP_0$, then $\{P_j\} \xrightarrow{j \rightarrow \infty} A^{-1}$ if $\|I - P_0A\|_2 = r < 1$.

Since, denoted by $E_j = I - P_jA$ we have that:

$$\|E_{j+1}\|_2 = \|I - 2P_jA + (P_jA)^2\|_2 = \|E_j^2\|_2 \leq \|E_j\|_2^2 \leq (r^{2^j})^2 = r^{2^{j+1}}$$

which implies $\lim_{j \rightarrow \infty} \|E_j\|_2 = 0$.

From Newton's method to a polynomial preconditioner for CG

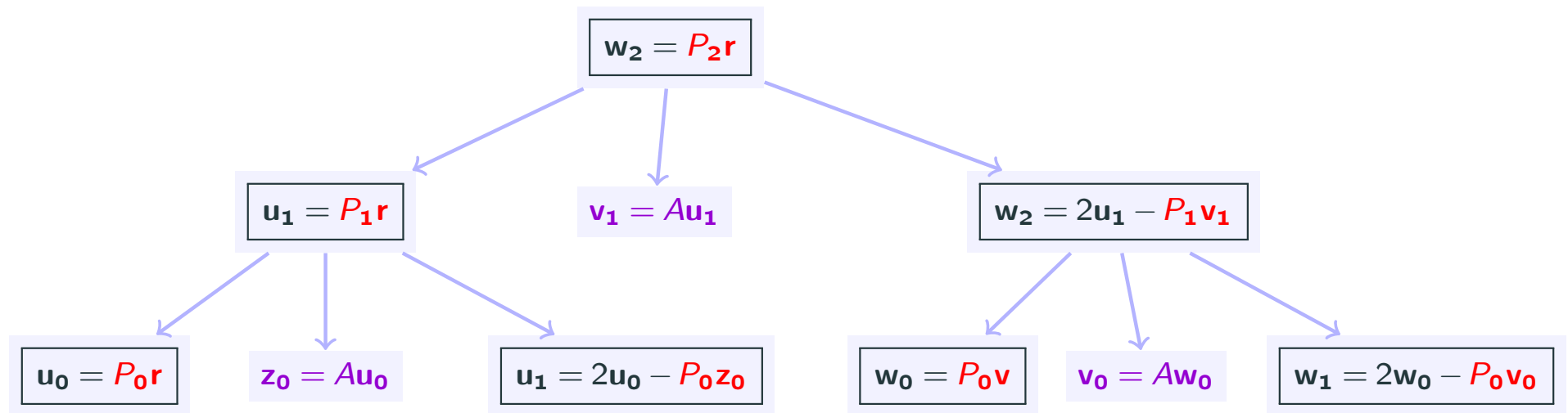
Given j , P_j is an approximate inverse of A and thus can be used as a preconditioner.

Matrices $\{P_j\}$ are not explicitly formed, the only thing that's needed within the Conjugate Gradient iteration is the computation of P_{j+1} times the residual vector.

This can be done recursively:
$$\mathbf{w} = P_{j+1}\mathbf{r} \iff \begin{cases} \mathbf{u} = P_j\mathbf{r} \\ \mathbf{v} = A\mathbf{u} \\ \mathbf{w} = 2\mathbf{u} - P_j\mathbf{v} \end{cases}$$

Preconditioner application entirely based on matrix-vector products with A and daxpys.

Example: For $j = 2$



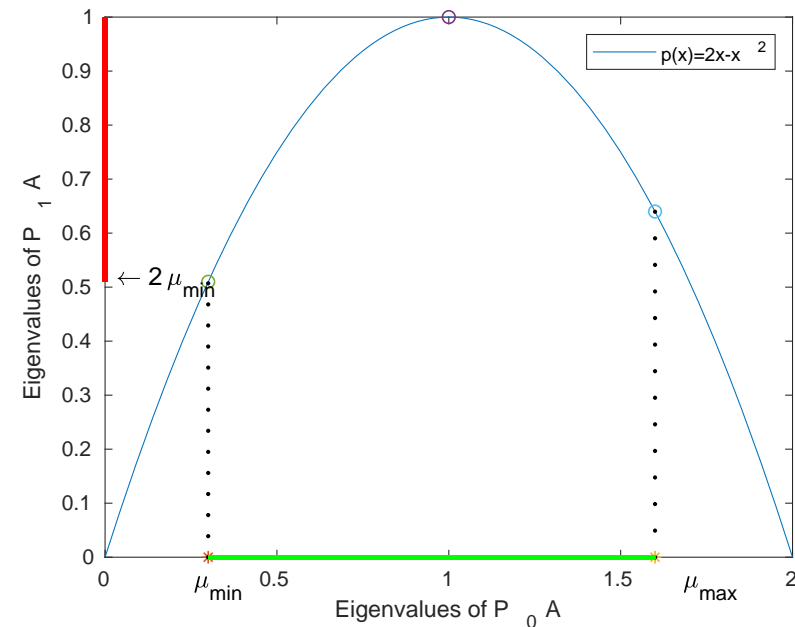
Question: Increasing j by 1 the cost of applying the preconditioner (roughly) doubles. Is it worth?

The condition $\|I - P_0A\|_2 < 1$, with P_0A symmetric, simplifies to $0 < \lambda(P_0A) < 2$.

The eigenvalues of $P_1A = 2P_0A - (P_0A)^2$ are $2\mu - \mu^2$ with $\mu \in \sigma(P_0A)$.

Assuming $1 \in \sigma(P_0A)$ we have that $\sigma(P_1A) \approx [2\mu_{\min}, 1]$ and so

$$\kappa(P_1A) > \frac{1}{2\mu_{\min}} > \frac{\kappa(P_0A)}{4}.$$



Further application of the recursion maps the eigenvalues of P_1A approximately into $[2\mu_1, 1]$ with $\mu_1 = \min_{\lambda \in \sigma(P_1A)} \lambda$ so $\kappa(P_2A) \approx \frac{\kappa(P_1A)}{2}$.

Condition number of preconditioned matrices halves \rightarrow PCG iterations decrease by a factor $\sqrt{2}$. So the answer to the question is **Newton's method is not worth.**

The efficiency of the Newton preconditioner can however be increased due to the following result:

Theorem

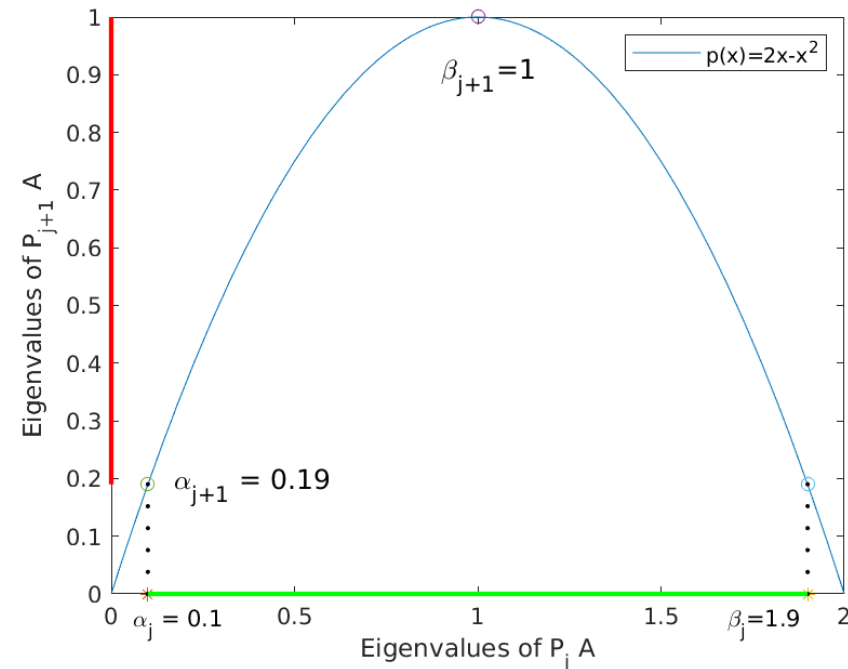
Let α_j, β_j be the smallest and the largest eigenvalues of $P_j A$.

If $0 < \alpha_j < 1 < \beta_j \leq 2 - \alpha_j$ then $[\alpha_{j+1}, \beta_{j+1}] \subset [2\alpha_j - \alpha_j^2, 1]$.

If $\beta_j = 2 - \alpha_j$ then the reduction in the condition number from $P_j A$ to $P_{j+1} A$ is near 4 provided that α_j is small:

$$\begin{aligned} \frac{\kappa(P_j A)}{\kappa(P_{j+1} A)} &= \frac{2 - \alpha_j}{\alpha_j} (2\alpha_j - \alpha_j^2) \\ &= (2 - \alpha_j)^2 \approx 4. \end{aligned}$$

Reduction of the condition number by a factor 4 \implies reduction of CG iterations by a factor 2.



We construct a sequence of scaling factors ζ_j so that **at each level j the eigenvalues of the preconditioned matrices $P_j A$ satisfy the hypothesis of the previous Theorem.**

At the first Newton stage the preconditioner must be scaled by $\zeta_0 = \frac{2}{\alpha_0 + \beta_0}$, so that

$$\begin{aligned} \alpha_0 &\longrightarrow \hat{\alpha}_0 = \frac{2\alpha_0}{\alpha_0 + \beta_0} \\ \beta_0 &\longrightarrow \hat{\beta}_0 = \frac{2\beta_0}{\alpha_0 + \beta_0} \end{aligned}$$

and

$$2 - \hat{\alpha}_0 = 2 - \frac{2\alpha_0}{\alpha_0 + \beta_0} = \frac{2\beta_0}{\alpha_0 + \beta_0} = \hat{\beta}_0.$$

Hence the eigenvalues of $P_1 A = (2\zeta_0 I - \zeta_0^2 A) A$ will lie in $[\alpha_1, \beta_1]$ where $\beta_1 = 1$ and

$\alpha_1 = (2 - \alpha_0 \zeta_0) \alpha_0 \zeta_0$ and the next scaling factor will be $\zeta_1 = \frac{2}{1 + \alpha_1}$.

The relation between two consecutive scaling factors ζ_{j+1} and ζ_j can be written as

$$\zeta_j = \frac{2}{1 + 2\zeta_{j-1} - \zeta_{j-1}^2}.$$

The final recurrence for the scaled Newton preconditioner is written as

$$\begin{aligned}P_0 &= \zeta_0 I \\P_{j+1} &= \zeta_{j+1} (2P_j - P_j A P_j), \quad j \geq 0\end{aligned}$$

and the corresponding recurrence between the polynomials p_{2^j-1} verifying $P_j = p_{2^j-1}(A)$

$$\begin{aligned}p_0(x) &= \zeta_0 \\p_{2^{j+1}-1}(x) &= \zeta_{j+1} \left(2p_{2^j-1}(x) - x p_{2^j-1}^2(x) \right), \quad j \geq 0.\end{aligned}$$

An analogous recurrence was developed in:



V. Pan and R. Schreiber,

An Improved Newton Iteration for the Generalized Inverse of a Matrix, with Applications,
SIAM Journal on Scientific and Statistical Computing, 1991

Algorithm 1 Newton-based polynomial preconditioner

- 1: Roughly approximate the extremal eigenvalues of A : α_0, β_0 .
- 2: Set the number of Newton steps: n_{lev}
- 3: Set

$$\zeta_0 = \frac{2}{\alpha_0 + \beta_0}, \quad \zeta_1 = \frac{2}{1 + 2\alpha_0\zeta_0 - (\alpha_0\zeta_0)^2},$$

$$\zeta_j = \frac{2}{1 + 2\zeta_{j-1} - \zeta_{j-1}^2}, \quad j = 2, \dots, n_{lev}.$$

- 4: Recursive application of P_j to a vector \mathbf{u} at each PCG iteration

$$P_0\mathbf{u} = \zeta_0\mathbf{u}$$

$$P_{j+1}\mathbf{u} = \zeta_{j+1}(2P_j\mathbf{u} - P_jAP_j\mathbf{u}), \quad j = n_{lev} - 1, \dots, 0$$

Relation with Chebyshev polynomials

The optimal polynomial preconditioner $p_k \in \Pi_k$ solves

$$\min_{p_k \in \Pi_k} \|I - p_k(A)A\|_2 = \min_{p_k \in \Pi_k} \max_{\lambda \in \sigma(A)} |1 - p_k(\lambda)\lambda| \leq \min_{\substack{q_{k+1} \in \Pi_{k+1} \\ q_{k+1}(0) = 1}} \max_{x \in [\alpha, \beta]} |q_{k+1}(x)|$$

with $q_{k+1}(x) = 1 - xp_k(x)$, and $\alpha = \lambda_{\min}(A), \beta = \lambda_{\max}(A)$.

As known, the solution to this last problem is given by a suitably shifted and scaled Chebyshev polynomial: $q_k(x) = \frac{T_k(x)}{\sigma_k}$.

Using the relation $T_{2k}(x) = 2T_k^2(x) - 1$, $k \geq 1$, involving the Chebyshev polynomials of the 1st kind we obtain:

$$q_{2k}(x) = \frac{1}{\sigma_{2k}} (2\sigma_k^2 q_k^2(x) - 1), \quad \text{where } \sigma = \frac{\beta_0 + \alpha_0}{\beta_0 - \alpha_0}, \quad \sigma_1 = \sigma, \quad \sigma_{2k} = 2\sigma_k^2 - 1.$$

from which the following recursion for p_k holds:

$$p_{2k-1}(x) = \frac{2\sigma_k^2}{\sigma_{2k}} (2p_{k-1}(x) - xp_{k-1}^2(x)), \quad k \geq 1, \quad p_0(x) = \frac{2}{\alpha_0 + \beta_0}. \quad (3)$$

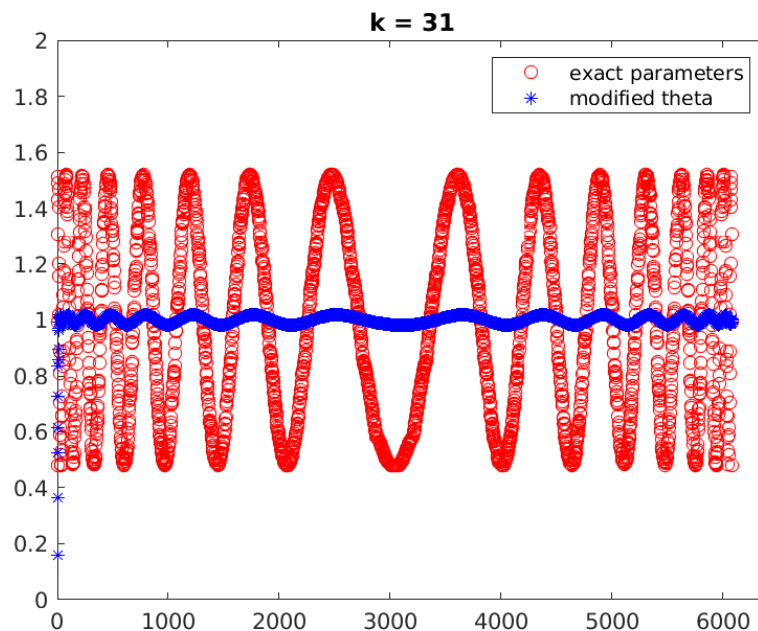
These polynomials are exactly the same as those generated by Newton's method.

The Newton-Chebyshev (NC) polynomial preconditioner is aimed at minimizing the condition number of $p_k(A)A$. However, this does not guarantee fast PCG convergence

The NC preconditioner is shown to cluster many eigenvalues around the smallest value.

We propose a simple modification of the algorithm to avoid this:

Scale ζ_0 as $\zeta_0 = (1 + \delta)\zeta_0$ with δ a small positive number.



Eigenvalue distribution of $p_k(A)A$ using the exact (red circles) and modified ζ_0 -value (blue stars) with $\delta = 0.01$, degree $k = 31$ on a FD matrix of dimension $n = 6084$

There are many red circles corresponding to the smallest eigenvalues. The blue stars are well separated. *We expect faster CG convergence with scaling!*

PCG iterations for solving the 78^2 ($n = 6084$) discretized Laplacian in the unit square with polynomial preconditioners of degree $k = 0, 1, 3, \dots, 31$, with $k = 2^j - 1$.

The extremal eigenvalues, the number ℓ of eigenvalues “close” to the minimum one and the condition number of the preconditioned matrices are also reported.

Original NC algorithm						NC with ζ_0 scaled by $1 + 10^{-2}$				
k	iter	μ_{\max}	μ_{\min}	ℓ	$\kappa(P_j A)$	iter	μ_{\max}	μ_{\min}	ℓ	$\kappa(P_j A)$
0	223	2.000	7.91e-04	1	2528.7	223	1.979	7.82e-04	1	2528.7
1	111	1.997	3.16e-03	2	632.7	112	1.958	3.06e-03	1	639.0
3	115	1.988	1.25e-02	188	158.7	61	1.849	1.13e-02	1	163.4
7	58	1.951	4.86e-02	278	40.2	31	1.564	3.52e-02	1	44.4
15	30	1.827	1.74e-01	468	10.5	17	1.189	8.22e-02	1	14.5
31	15	1.519	4.81e-01	874	3.2	11	1.018	1.60e-01	1	6.3

Smaller number of iterations with the scaled NC preconditioner despite of the (slightly) increasing condition number!

Numerical Results

Matrices and their characteristics:

matrix	n	nnz
Opt_Trans ¹	412417	2 882817
Lap1600	2 553604	12 761628
Emilia-923 ²	923136	41 005206

We performed a symmetric diagonal scaling of the original linear systems:

Define $D = \text{diag}(A)$ and solve $D^{-1/2}AD^{-1/2}\hat{\mathbf{x}} = D^{-1/2}\mathbf{b}$, $\mathbf{x} = D^{1/2}\hat{\mathbf{x}}$.

A rough approximation of the extremal eigenvalues was computed by the unpreconditioned DACG algorithm (with 0.01 relative tolerance for the exit test).



L. Bergamaschi, G. Gambolati, G. Pini.

Asymptotic Convergence of Conjugate Gradient Methods for the Partial Symmetric Eigenproblem
NLAA, 1997

The scaling factor was set to $1 + \delta$, $\delta = 10^{-3}$.

¹arises from the FE discretization of a continuous formulation of an optimal transport problem

²arises from the regional geomechanical model of a deep hydrocarbon reservoir. It is obtained discretizing the structural problem with tetrahedral Finite Elements

Number of PCG iterations and CPU times for matrices Opt_Transp and Lap1600 with NC polynomial preconditioner for various degrees ($k = 2^j - 1$).

k	Matrix Opt_Transp				Matrix Lap1600	
	iter	ddot	$A \times \mathbf{v}$	CPU(s)	iter	CPU(s)
0	3433	10299	3433	26.39	4517	203.52
1	1773	5319	3536	23.25	2313	176.58
3	879	2637	3516	20.95	1174	160.50
7	439	1317	3512	19.86	589	151.79
15	222	666	3552	19.59	295	147.83
31	117	351	3744	20.33	149	146.80
63	69	207	4416	23.85	77	151.17

1. The assessment of the extremal eigenvalues is relatively cheap (it took only 0.69 seconds for the Opt_Transp matrix and 1.33 seconds for the Laplacian).
2. Impressive reduction in the number of dot products (inner products and norms) and increasing number of matrix vector products.
3. The CPU time decreases up to a certain level only.

Numerical Results on Parallel computing environments

HPC Cluster Marconi located at CINECA (Bologna) with 3600 nodes and 1×68 -cores Intel Xeon 7250 CPU (Knights Landing) at 1.4GHz. 128 GB RAM per node and a 100 Gb/s fat-tree network.

Scalability analysis for matrix Emilia-923 ($n = 923136$, $nnz = 41005206$).

p	j = 5			j = 2			j = 0		
	iter	T_p	$E_p^{(16)}$	iter	T_p	$E_p^{(16)}$	iter	T_p	$E_p^{(16)}$
16	379	114.44		3008	115.64		11386	117.15	
64	379	33.33	86%	3008	34.43	84%	11382	37.74	78%
256	379	10.39	69%	3008	12.35	59%	11380	16.75	44%
512	379	6.15	58%	3008	9.15	39%	11380	14.70	25%

With $p = 512$ processors the total time reduces from 14.70 seconds (no preconditioning) to 6.15 seconds due to the great reduction in the number of dot products.

Due to the savings in global communication and synchronization the parallel efficiency increases from 25% ($j = 0$) to 58% ($j = 5$).

MARCONI 100: accelerated cluster based on 980 Nodes, each equipped with 2x16 cores IBM POWER9 AC922 at 3.1 GHz. Each node has 128GB RAM.

Matrices FD3d discretization of the Laplace equation on a cube with nx subdivision in each dimension.

nx	n	nnz	$\kappa(A)$
512	1.3×10^8	9.4×10^9	1.06×10^5
1024	1.1×10^9	7.5×10^9	4.24×10^5
2148	8.6×10^9	6.0×10^{10}	1.70×10^6

nx	p	$j = 5$			$j = 0$		Gain $T_p^{(0)} / T_p^{(5)}$
		iter	MVP	T_p	iter	T_p	
512	64	45	1440	67.0	1300	95.3	1.42
	128	45		36.2	1300	50.2	1.39
	256	45		21.8	1300	27.7	1.27
	512	45		13.8	1300	16.8	1.22
1024	64	88	2816	858.4	2553	1481.7	1.73
	256	88		254.3	2553	400.6	1.58
	1024	88		97.2	2553	131.5	1.35
2048	512	165	5280	1925.7	5033	3169.8	1.65
	2048	165		710.5	5033	1001.5	1.41

Comparison with the FSAI-AMG (Algebraic Multigrid) Method from [chronos](https://www.m3eweb.it/chronos/) package available at <https://www.m3eweb.it/chronos/>

Results on the FD matrix with $n_x = 512$.

p	AMG preconditioner with FSAI as smoother				FSAI preconditioner				NC prec. j=5	
	iter	T_{setup}	T_{solver}	T_p	iter	T_{setup}	T_{solver}	T_p	iter	T_p
64	23	21.5	10.8	32.3	786	5.9	86.4	92.4	45	67.0
128	24	15.3	7.1	22.4	774	2.7	43.7	46.5	45	36.2
256	26	12.4	4.4	16.8	782	1.7	24.4	26.2	45	21.8
512	28	14.3	4.6	18.8	758	0.7	13.8	14.5	45	13.8

Timings and scalability are comparable.

However the polynomial preconditioner enjoys the following desirable properties:

- It is matrix-free
- It does not require additional memory
- It greatly reduces the number of global communication/synchronization required in inner product computation, thus enhancing parallel efficiency in large scale parallel computing environments.

The NC polynomial preconditioner can be used with high degree to accelerate the PCG solution of large and sparse SPD linear systems. It requires only a rough approximation of the extremal eigenvalues.

Further convergence acceleration can be obtained by suitably scaling one parameter to avoid clustering of the smallest eigenvalues.

The NC polynomial preconditioner reduces the number of PCG iterations (reduces the number of dot products) by packing more work (matrix-vector products) into each iteration.

In large scale parallel computing environments the NC polynomial preconditioner is a collective-communication avoiding technique that improves scalability and parallel performance.

Work in progress

Optimal selection of the δ scaling parameter

Using the NC polynomial preconditioner as a second-level preconditioner



L. Bergamaschi, and A. Martínez

Parallel Newton-Chebyshev polynomial preconditioners for the Conjugate Gradient method
Computational and Mathematical Methods, 2021



M. Embree, J. Loe and R. B. Morgan,

Polynomial Preconditioned Arnoldi with Stability Control
SIAM Journal on Scientific Computing, 2021



M. Embree, and J. Loe,

Toward Efficient and Stable Polynomial Preconditioning for GMRES
arXiv:1911.07065 [math.NA]



Q. Liu, R.B. Morgan, W. Wilcox

Polynomial preconditioned GMRES and GMRES-DR
SIAM Journal on Scientific Computing, 2015



J. Loe, H. K. Thornquist, E. G. Boman

Polynomial Preconditioned GMRES to Reduce Communication in Parallel Computing
arXiv:1907.00072 [math.NA]



X. Ye, Y. Xi, and Y. Saad.

Preconditioning via GMRES in polynomial space.
Preprint ys-2019-03, University of Minnesota, Minneapolis, MN.



**Thank you so much
for your attention !**