A Fast Parameter-Free Preconditioner for Structured Grid Problems

Abhinav Aggarwal
Google London
Email: abhinavagg96@gmail.com

Sivam Kakkar
CSTAR, IIIT,H, Hyderabad
Email: shivam543@gmail.com

Pawan Kumar
CSTAR, IIIT,H, Hyderabad
Email: pawan.kumar@iiit.ac.in

***Abstract*—A fast, robust, parallel, and parameter free version of a frequency filtering preconditioner is proposed for linear systems corresponding to diffusion equation on a structured grid. Proposed solver is faster than the state-of-the-art solvers.**

***Index Terms*—Multithreading, Large Sparse Linear System, Conjugate Gradient Method, Preconditioner**

I. Introduction

We consider the problem of solving large sparse linear systems of the form

$$Ax = b, \quad A \in \mathbb{R}^{m\times m}, \quad b \in \mathbb{R}^m, \tag{1}$$

which arises, for example, during the numerical solution of the following diffusion equation

$$\begin{aligned} -\text{div}(\kappa(x)\nabla u) &= f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega_D, \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on } \partial\Omega_N. \end{aligned} \tag{2}$$

Here Ω is the interior of the domain, and $\partial\Omega_D$ and $\partial\Omega_N$ are the Dirichlet and Neumann boundaries respectively. Such problems appear as subproblems to wide variety of numerical simulations in fluid dynamics, material science, etc. On a structured grid, the discretization schemes such as finite difference, finite element, etc. lead to a "nested" block tridiagonal matrix (Fig. 2), which is symmetric and positive definite.

II. The Proposed Filtering Preconditioner

Let the plane block (three big diagonal blocks in Fig. 2) be denoted by $\hat{D}_i$, the line blocks by $\overline{D}_i$, (9 red diagonal blocks in Fig. 2) and the cell blocks by $\tilde{D}_i$. Similarly, let us denote the interface unknowns for plane blocks be denoted by $\hat{L}_i, \hat{U}_i$, line blocks by $\overline{L}_i, \overline{U}_i$, and so on. The blocks are twisted around middle block row as shown in middle figure in Fig. 2, where an upper triangular blocks in blue becomes lower triangular blocks after middle diagonal red block resulting in a "twist".

A. Construction of Twisted Filtering Preconditioner

To expose parallelism in the proposed Filtering Decomposition (called NTD), In actual implementation, we only need to store the bands of A. To create the preconditioner, we first consider the block LU factorization $A = (P + L_3)(I + P^{-1}U_3)$. The A in this equation is already known to us, and on simplifying the right hand side, and solving for diagonal blocks P_i of P, we get the following recurrence solution for P_i

$$P_i = \begin{cases} \widehat{D}_1, & i = 1, \\ \widehat{D}_i - \widehat{L}_3^{i-1}(P_{i-1}^{-1})\widehat{U}_3^{i-1}, & i = 2, \cdots, j-1, \\ \widehat{D}_{nz}, & i = nz, \\ \widehat{D}_i - \widehat{L}_3^{i}(P_{i+1}^{-1})\widehat{U}_3^{i}, & i = nz-1, \cdots, j+1, \\ \widehat{D}_i - \widehat{L}_3^{i-1}(P_{i-1}^{-1})\widehat{U}_3^{i-1} - \widehat{L}_3^{i}(P_{i+1}^{-1})\widehat{U}_3^{i}, & i = j. \end{cases} \tag{3}$$

In the above iteration, as i increases, P_i tends to become denser, hence, it is costly to compute terms such as $\widehat{L}_3^{i-1}(P_{i-1}^{-1})\widehat{U}_3^{i-1}$. Moreover, storing P_i is costly, hence, we will replace P_i^{-1} by its sparse approximation. Reusing the notation P_i for approximated P_i, we define the following approximation to P_i

$$P_i = \begin{cases} \widehat{D}_1, & i = 1, \\ \widehat{D}_i - \widehat{L}_3^{i-1}(2\beta_{i-1} - \beta_{i-1}P_{i-1}\beta_{i-1})\widehat{U}_3^{i-1}, \\ \quad i = 2, \cdots, j-1, \\ \widehat{D}_{nz}, & i = nz, \\ \widehat{D}_i - \widehat{L}_3^{i}(2\beta_{i+1} - \beta_{i+1}P_{i+1}\beta_{i+1})\widehat{U}_3^{i}, \\ \quad i = nz-1, \cdots, j+1, \\ \widehat{D}_i - \widehat{L}_3^{i-1}(2\beta_{i-1} - \beta_{i-1}P_{i-1}\beta_{i-1})\widehat{U}_3^{i-1} \\ \quad -\widehat{L}_3^{i}(2\beta_{i+1} - \beta_{i+1}P_{i+1}\beta_{i+1})\widehat{U}_3^{i}, \quad i = j. \end{cases} \tag{4}$$

Here j is the block row index where the twist happens, β_i are diagonal matrices defined as

$$\beta_i = \text{diag}((P_{i-1}^{-1}\hat{U}^{i-1})./(\hat{U}^{i-1}\hat{t}_i)),$$

where $\hat{t}_i$ is a vector of all ones, and $2\beta_{i-1} - \beta_{i-1}P_{i-1}\beta_{i-1}$, or $2\beta_{i+1} - \beta_{i+1}P_{i+1}\beta_{i+1}$ for the lower half is claimed to be a better approximation to $(P_i)^{-1}$. Note that the product on the rhs no longer equals A after substituting P^{-1} with it's β approximated form. After approximation, we define the NTD preconditiner B_{NTD} as follows

$$B_{\text{NTD}} = (P + L_3)(I + P^{-1}U_3). \tag{5}$$

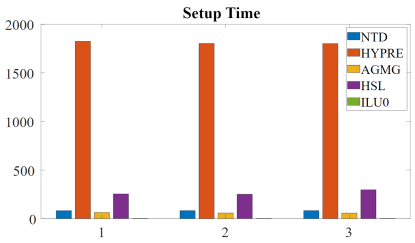
Since $\beta_i's$ are diagonals, the sparsity pattern of P_i is same as that of $\widehat{D}_i$. Hence, like $\hat{D}_i$ blocks, the individual P_i blocks are themselves nested block tridiagonal, we can obtain a further factorization as follows

$$P = (T + L_2)(I + T^{-1}U_2). \tag{6}$$

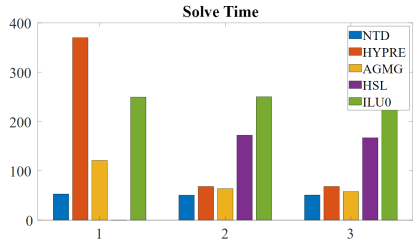
As for P_i blocks before, we have the following recurrence solution for T_i blocks

$$T_i = \begin{cases} \overline{D}_1, & i = 1, \\ \overline{D}_i - \overline{L}_2^{i-1}(2\beta_{i-1} - \beta_{i-1}T_{i-1}\beta_{i-1})\overline{U}_2^{i-1}, \\ \quad i = 2, \cdots, j-1, \\ \overline{D}_{nz}, & i = nz, \\ \overline{D}_i - \overline{L}_2^{i}(2\beta_{i+1} - \beta_{i+1}T_{i+1}\beta_{i+1})\overline{U}_2^{i}, \\ \quad i = nz-1, \cdots, j+1, \\ \overline{D}_i - \overline{L}_2^{i-1}(2\beta_{i-1} - \beta_{i-1}T_{i-1}\beta_{i-1})\overline{U}_2^{i-1} \\ \quad -\overline{L}_2^{i}(2\beta_{i+1} - \beta_{i+1}T_{i+1}\beta_{i+1})\overline{U}_2^{i}, \quad i = j, \end{cases} \tag{7}$$

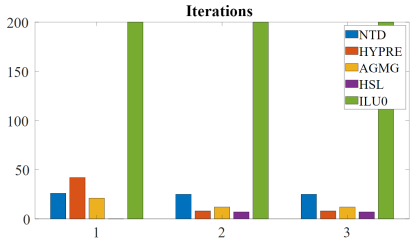
where j is the block row index, $\beta_i's$ are diagonal matrices defined as $\beta_i = \text{diag}((T_{i-1}^{-1}\overline{U}^{i-1})./(\overline{U}^{i-1}\bar{t}_i))$, where $\bar{t}_i$ is vector of all ones, and as shown above, we consider the approximation $2\beta_{i-1} - \beta_{i-1}T_{i-1}\beta_{i-1}$ for upper half, and similarly the approximation $2\beta_{i+1} - \beta_{i+1}T_{i+1}\beta_{i+1}$ for the lower half is claimed to be a better approximation to $(T_i)^{-1}$. Again sparsity pattern of T_i is same as $\bar{D}_i$, i.e., the T_i blocks



(a) Horizontal axis: Problem type. Vertical Axis: Setup time in sec.

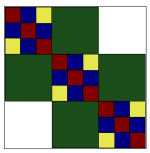


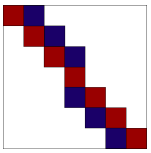
(b) Horizontal axis: Problem Type. Vertical Axis: Solve time in seconds.



(c) Horizontal axis: Problem Type. Vertical Axis: Iterations.

Fig. 1: NTD: proposed solver, AGMG: Aggregation based multigrid [4], Hypre [3], HSL [2].





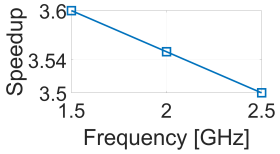


Fig. 2: Left: Nested tridiagonal structure. Middle: Twisted block upper triangular matrix. Right: Speedup on 4 cores.

are themselves pointwise tridiagonal matrices, and can be approximated similarly as follows:

$$T = (M + L_1)(I + M^{-1}U_1). \tag{8}$$

Since T is block diagonal with tridiagonal blocks, the above factorization is exact. We obtain the recurrence for M_i as follows:

$$M_i = \begin{cases} \tilde{D}_1, & i = 1, \\ \tilde{D}_i - \tilde{L}_1^{i-1} M_{i-1}^{-1} \tilde{U}_1^{i-1}, & i = 2, \cdots, j-1, \\ \tilde{D}_{nz}, & i = nz, \\ \tilde{D}_i - \tilde{L}_1^{i} M_{i+1}^{-1} \tilde{U}_1^{i}, & i = nz-1, \cdots, j+1, \\ \tilde{D}_i - \tilde{L}_1^{i-1} M_{i-1}^{-1} \tilde{U}_1^{i-1} - \tilde{L}_1^{i} M_{i+1}^{-1} \tilde{U}_1^{i}, & i = j, \end{cases} \tag{9}$$

where j is the row index, and M_{i-1}^{-1} (or M_{i+1}^{-1}) is reciprocal of M_{i-1} (or M_{i+1}). Note that during construction of the preconditioner, we only need to store the bands as vectors with proper padding by zeros. Also, note that to extract these bands, we do not construct the matrices T, P, and B. We only extract these bands during the recurrence for T_i and P_i. Also, the outermost bands ℓ_3 and u_3 are same as the outermost bands of A. We note that due to twists, there is two way parallelism in computing P_i, T_i, and M_i, because recursions (4), (7), and (9) run from both ends, hence, a total of 8-way parallelism. We found it more efficient to use SIMD operations for recurrence for M_i, see [1] for details. Due to the twists, solves and setup can now be done in parallel. For more parallelism, the solver can be used as a subdomain solver inside additive Schwarz, or other structured domain decomposition solver. This is an ongoing research.

III. Experimental Details

Experiments ran on intel i7-7700K CPU with 4 physical cores, 64 GB DDR4 RAM; ompiler version is `gcc 7.3` with `-march=native` and `-O3` flags. We consider Type-1, Type-2, and Type-3 matrices corresponding to diffusion problem with jumps of order 10^3, poisson, and non-homogeneous problems [1]. We use incomplete LU with no-fill as a smoother, and combine it with NTD preconditioner described previously. The combination preconditioner denoted by B_c can be defined as follows:

$$B_c^{-1} = B_{\text{NTD}}^{-1} + B_{\text{ILU0}}^{-1} - B_{\text{NTD}}^{-1} A B_{\text{ILU0}}^{-1}, \tag{10}$$

where B_{ILU0} denotes the ILU preconditioner [5]. From the equation (10) above, we notice that solving with the preconditioner B_c requires solving with B_{NTD} and B_{ILU0}, and a matrix vector multiplication with the given coefficient matrix A. The solve with B_{ILU0} requires triangular solves, i.e. forward sweep followed by a backward sweep, which are inherently sequential in nature. To address this bottleneck, instead of doing ILU0 for full matrix A, we do an ILU0 of a block diagonal approximation $\tilde{A}$ of matrix A, where

$$\tilde{A} = \text{blkD}(A(1:M, 1:M), A(M+1:N, M+1:N)),$$

where $M = \text{floor}(N/2)$, and blkD stands for block diagonal. We use two threads during the construction and solve phase of B_{ILU0}. For the matrix times vector operation, we have implemented a parallel banded matrix vector multiply routine engaging 4 threads, further leveraging SIMD operations inside each thread [1]. Storage cost for NTD is same as A, moreover, since ILU0 is also used, the total memory required for the preconditioner is twice that of A, which is comparable to memory requirements for multigrid solvers. All the solvers were used with CG, max iterations was 200, and tolerance for the decrease of relative residual was 10^{-7}. For Hypre [3], coarsen type is 3, relax type is 6, strong threshhold 0.5, aggregation number of levels, interpolation type, and truncation factor is 0. For HSL [2], coarse solver is 2, smoother is 2, V-cycles is 2. For AGMG [4], Gauss-Siedel smoother is used, other parameters are default.

IV. Numerical Experiments

The number of unknowns of the linear system is 43 Million. The Fig. 1 show clearly that the proposed solver NTD is fastest solve times for all types of the problems. In setup times, it is as fast as AGMG. In total time, it is fastest on type-1, and comparable to AGMG for type-2 and type-3 problems. Also, the iterations are plotted in Fig 3. Note that bar plots for setup and solve times corresponding to HSL for type-1 matrix is missing due to an error thrown when running. Also, compared to other methods, our method does not require parameter tuning. We choose best parameters for others using grid search. The scalability of the proposed method for variations in CPU frequency is shown in Fig. 2. We observe that speedup on 4 cores remains between 3.5 and 3.6. For results with various problem sizes, see [1].

REFERENCES

[1] A. Aggarwal, S. Kakkar, P. Kumar, Multithreaded Filtering Preconditioner for Diffusion Equation on Structured Grid, arXiv 1909.09771v1, 2020.

[2] HSL_MI20 Unsymmetric system: algebraic multigrid preconditioner, https://www.hsl.rl.ac.uk/catalogue/hsl_mi20.html

[3] HYPRE: Scalable Linear Solvers and Multigrid Methods, https://github.com/hypre-space/hypre.

[4] AGMG: Iterative solution with AGgregation-based algebraic MultiGrid, http://agmg.eu/

[5] Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS publishing company, Boston, MA, 1996.