A Fast Parameter-Free Preconditioner for Structured Grid Problems

Abhinav Aggarwal, Shivam Kakkar, Pawan Kumar
Google, London and IIIT, Hyderabad
Department of Computer Science

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

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

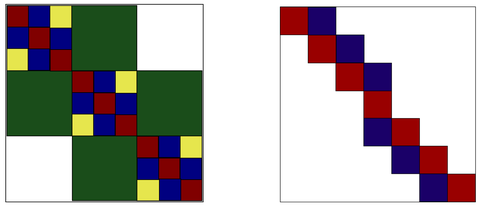
- Ω is the interior of the domain, and $\partial\Omega_D$ and $\partial\Omega_N$ are the Dirichlet and Neumann boundaries
- It appear as subproblem in fluid dynamics, material science, etc.
- On structured grid, discretization schemes lead to “nested" block tridiagonal matrix.
- Exploiting the structure can lead to scalable and memory efficient solver. For solvers, see [1]

Objectives

To design fast, memory efficient, and parameter-free linear solver for structured linear system.

Proposed Method (NTD)

- For nested tridiagonal structure, data corresponding to discrete points on the plane is denoted by matrix $\hat{D}_i$, the data corresponding to the points on lines by block $\overline{D}_i$, and the data for points on the line by $\tilde{D}_i$.



- Denote the edge weights between planes i and $i+1$ by $\hat{L}_i, \hat{U}_i$, (in green) similarly, for line by $\overline{L}_i, \overline{U}_i$, (in blue) and so on.
- In right figure above, upper diagonal blocks in blue become lower diagonal blocks reflecting a "twist". See [1].
- The three lower, upper blocks are twisted, which creates parallelism.
- For preconditioner, consider the block LU factorization $A = (P + L_3)(I + P^{-1}U_3)$. L_3 is matrix with lowermost band, similarly U_3.
- Solving for diagonal blocks P_i of P, we get the following

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

- P_i becomes denser, hence, costly to compute $\widehat{L}_3^{i-1}(P_{i-1}^{-1})\widehat{U}_3^{i-1}$.

The Proposed Method Continued

- Storing P_i is costly as well, approximate P_i^{-1} above:

$$P_i = \begin{cases} \widehat{D}_1, \quad i = 1, \\ \widehat{D}_i - \widehat{L}_3^{i-1}(2\beta_{i-1} - \beta_{i-1}P_{i-1}\beta_{i-1})\widehat{U}_3^{i-1}, \\ \quad i = 2, \cdots, j-1, \\ \widehat{D}_{nz}, \quad i = nz, \\ \widehat{D}_i - \widehat{L}_3^{i}(2\beta_{i+1} - \beta_{i+1}P_{i+1}\beta_{i+1})\widehat{U}_3^{i}, \\ \quad i = nz-1, \cdots, j+1, \\ \widehat{D}_i - \widehat{L}_3^{i-1}(2\beta_{i-1} - \beta_{i-1}P_{i-1}\beta_{i-1})\widehat{U}_3^{i-1} \\ \quad -\widehat{L}_3^{i}(2\beta_{i+1} - \beta_{i+1}P_{i+1}\beta_{i+1})\widehat{U}_3^{i}, \quad i = j. \end{cases} \tag{5}$$

- In recursion (5), twist happens around jth block row.
- 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.
- $2\beta_{i-1} - \beta_{i-1}P_{i-1}\beta_{i-1}$, or $2\beta_{i+1} - \beta_{i+1}P_{i+1}\beta_{i+1}$ in (5) for the lower half is claimed to be a better approximation to $(P_i)^{-1}$.
- With the diagonal blocks P_i of P approximated by (5), define the proposed preconditioner denoted by B_{NTD} as follows

$$B_{\text{NTD}} = (P + L_3)(I + P^{-1}U_3). \tag{6}$$

- Since $\beta_i's$ are diagonals, sparsity pattern of P_i same as $\widehat{D}_i$.
- Hence, like $\hat{D}_i$ blocks, P_i are also nested block tridiagonal
- Hence as for A, we can obtain a further factorization as follows

$$P = (T + L_2)(I + T^{-1}U_2). \tag{7}$$

- As for P_i before, we have the following recurrence for T_i

$$T_i = \begin{cases} \overline{D}_1, \quad i = 1, \\ \overline{D}_i - \overline{L}_2^{i-1}(2\beta_{i-1} - \beta_{i-1}T_{i-1}\beta_{i-1})\overline{U}_2^{i-1}, \\ \quad i = 2, \cdots, j-1, \\ \overline{D}_{nz}, \quad i = nz, \\ \overline{D}_i - \overline{L}_2^{i}(2\beta_{i+1} - \beta_{i+1}T_{i+1}\beta_{i+1})\overline{U}_2^{i}, \\ \quad i = nz-1, \cdots, j+1, \\ \overline{D}_i - \overline{L}_2^{i-1}(2\beta_{i-1} - \beta_{i-1}T_{i-1}\beta_{i-1})\overline{U}_2^{i-1} \\ \quad -\overline{L}_2^{i}(2\beta_{i+1} - \beta_{i+1}T_{i+1}\beta_{i+1})\overline{U}_2^{i}, \quad i = j, \end{cases} \tag{8}$$

- $\beta_i's$ are diagonal matrices defined as $\beta_i = \text{diag}((T_{i-1}^{-1}\overline{U}^{i-1})./(\overline{U}^{i-1}\bar{t}_i))$.
  - $\bar{t}_i$ is vector of all ones.
- Approximate $(T_i)^{-1}$ by $2\beta_{i-1} - \beta_{i-1}T_{i-1}\beta_{i-1}$ for upper half.
- Similarly approximate $(T_{i+2})^{-1}$ by $2\beta_{i+1} - \beta_{i+1}T_{i+1}\beta_{i+1}$.
- T_i blocks are pointwise tridiagonal, can be approximated:

$$T = (M + L_1)(I + M^{-1}U_1). \tag{9}$$

- T is block diagonal with tridiagonal blocks, (9) is exact.
- Since A is structured and 7-banded, only need to store bands.
- Whole preconditioner NTD requires 6 bands storage (less than A).
- Operations during constructions of P, T are mostly vector operations.
- Due to twists, recursions (5), (8), (9) can start from top and bottom.
  - Hence, 8-way parallelism is achieved. See [1] for detail.

Graphical Results

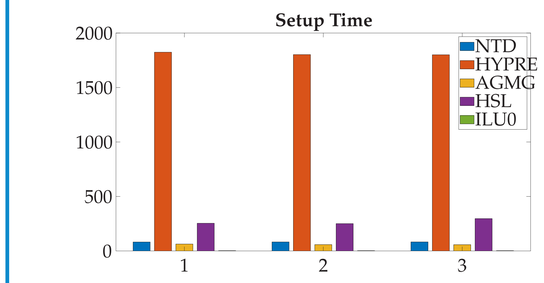


Figure 1: Comparison of setup times for various solvers. Horizontal axis: Problem type; Vertical axis: Setup time in seconds.

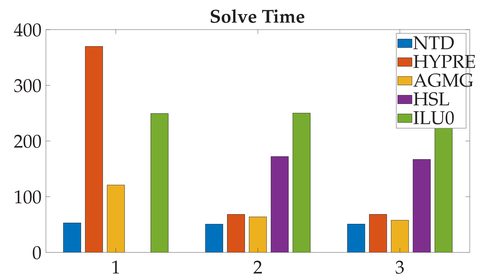


Figure 2: Comparison of solve times for various solvers. Horizontal axis: Problem type; Vertical axis: Solve time in seconds.

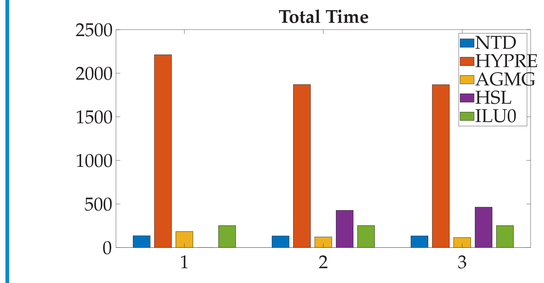


Figure 3: Comparison of total time (setup+solve) for various solvers. Horizontal axis: Problem type; Vertical axis: Total time in seconds.

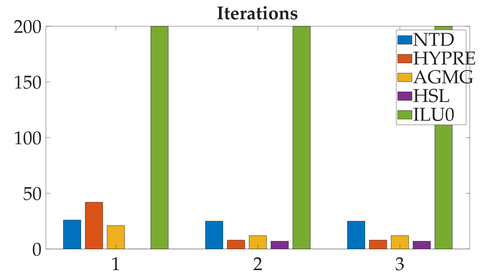


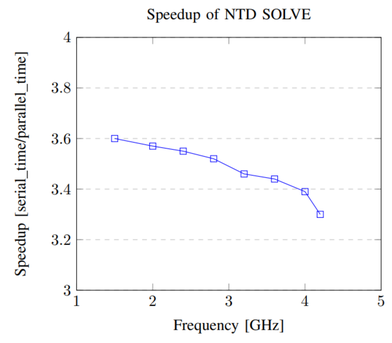
Figure 4: Comparison of iterations for various solvers. Horizontal axis: Problem type; Vertical axis: Iterations

Numerical Experiments

Experiments ran on intel i7-7700K CPU with 4 physical cores, 64 GB DDR4 RAM. The compiler version used is `gcc 7.3` with `-march=native` and `-O3` flags. We consider Type-1, Type-2, and Type-3 corresponds to diffusion with checkerboard jumps of order 1e3, Poisson, and non-homogeneous problems [1].
As in multigrid, we use incomplete LU factorization of given matrix A with no fill-in as a smoother and combine it with NTD preconditioner described previously. The combination preconditioner denoted by B_c can be defined as follows:

$$B_c^{-1} = B_{\text{NTD}}^{-1} + B_{\text{ILU0}}^{-1} - B_{\text{NTD}}^{-1}AB_{\text{ILU0}}^{-1}, \tag{3}$$

where B_{ILU0} denotes the ILU preconditioner [2]. From the equation (3) 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. We have showed how to solve with B_{NTD} on a quad core in previous sections. 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. Total storage cost for proposed method is approximately twice that of A.



- Such an approximation does not lead to any degradation in the performance of the combination preconditioner, and allowed us to engage two threads during the construction and solve phase of 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. Please refer to [1].
- **Results:** Figures 1 and 2 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 4. Note that bar corresponding to HSL for type-1 matrix is missing, because the code error for this method for this type. Also, compared to the methods, our method does not require parameter tuning. For other methods, we choose best parameters. We also show speedup of NTD as a function of CPU frequency. For frequency above 3, we achieve speedup above 3.4.

For detailed analysis of experimental results and the state-of-the-art solvers, see the detailed report [1].

Ongoing Research

The ongoing research is on testing the proposed solver as subdomain solver in structured domain decomposition methods.

References

[1] A. Aggarwal, S. Kakkar, P. Kumar, Multithreaded Filtering Preconditioner for Diffusion Equation on Structured Grid, arXiv 1909.09771v1, 2020
[2] Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS publishing company, Boston, MA, 1996.