A GPU-Friendly SSOR Preconditioner Based on an Approximate Inverse of Triangular Matrices

Yuya Kudo
Hokkaido University
Sapporo, Japan
kudo.yuya.r7@elms.hokudai.ac.jp

Yuki Satake
Hokkaido University
Sapporo, Japan
(Advisor)

Takeshi Fukaya
Hokkaido University
Sapporo, Japan
(Advisor)

Takeshi Iwashita
Kyoto University
Kyoto, Japan
(Advisor)

Abstract

In this study, we develop a GPU-friendly variant of the symmetric successive over-relaxation (SSOR) preconditioner for efficient execution of the preconditioned conjugate gradient (PCG) method on GPUs. To avoid the forward and backward substitutions required by the original SSOR preconditioner, which constitute a major bottleneck on GPUs, we incorporate an approximate inverse of a triangular matrix appearing in the SSOR preconditioner. We further introduce a dropping strategy to reduce the computational cost of applying the approximate inverse. We demonstrate the effectiveness of the proposed SSOR variant through comparisons with existing parallel SSOR variants on an NVIDIA H100 GPU.

1 Introduction

In many scientific and engineering fields, such as structural analysis and electromagnetics, systems of linear equations ($A\boldsymbol{x} = \boldsymbol{b}$) with a large, sparse, symmetric positive definite (SPD) coefficient matrix $A \in \mathbb{R}^{n\times n}$ frequently arise and are commonly solved using the preconditioned conjugate gradient (PCG) method. Recently, computing platforms have increasingly shifted toward GPU-centric architectures, requiring numerical methods to be adapted for efficient execution on GPUs. While the main computational kernels of the CG method, such as sparse matrix–vector multiplication (SpMV) and vector operations, can be efficiently executed on GPUs, many existing preconditioning techniques are difficult to implement efficiently on GPUs. Therefore, developing GPU-friendly preconditioners is one of the key challenges in realizing efficient GPU-based linear solvers.

This study focuses on the symmetric successive over-relaxation (SSOR) preconditioner, a widely used preconditioner for the PCG method. Although the SSOR preconditioner can improve the convergence of the CG method for many practical problems, it is difficult to implement efficiently on GPUs because its main computational kernels are forward and backward substitutions, which are inherently sequential. To address this issue, motivated by recent promising results on approximate-inverse-based preconditioners (e.g., [5]), we propose incorporating an approximate inverse of a triangular matrix into the SSOR preconditioner. Based on this idea, we develop a GPU-friendly variant of the SSOR preconditioner together with an efficient construction of the approximate inverse and demonstrate its effectiveness through comparisons with existing parallel SSOR preconditioners.

2 The SSOR preconditioner

The SSOR preconditioner [4] is given by

$$M_{\mathrm{SSOR}} = \frac{1}{\omega(2-\omega)}(D+\omega L)D^{-1}(D+\omega L)^{\mathrm{T}}.$$

Here, ω ($0 < \omega < 2$) is the relaxation parameter, L is the strictly lower triangular part of A, and D is the diagonal part of A.

When applying the SSOR preconditioner to the CG method, we solve $M_{\mathrm{SSOR}}\boldsymbol{z} = \boldsymbol{r}$, where $\boldsymbol{r}$ is the residual vector, in the following two steps:

(1) Forward substitution: $\boldsymbol{y} = \omega(2-\omega)(D+\omega L)^{-1}\boldsymbol{r}$
(2) Backward substitution: $\boldsymbol{z} = (D+\omega L)^{-\mathrm{T}}D\boldsymbol{y}$

However, both substitution steps are inherently sequential, making them difficult to implement efficiently on GPUs.

3 Proposed method

The proposed method incorporates an approximate inverse of the triangular matrix $D+\omega L$ into the SSOR preconditioner. Specifically, a lower triangular matrix W satisfying

$$(D+\omega L)W \approx I$$

is computed, where I denotes the identity matrix. While $(D+\omega L)^{-1}$ is lower triangular, it is generally dense. Therefore, to limit computational and storage costs, we restrict W to have the same sparsity pattern as $D+\omega L$. The application of the resulting SSOR variant consists of the following two SpMV operations:

(1) $\boldsymbol{y} = \omega(2-\omega)W\boldsymbol{r}$,
(2) $\boldsymbol{z} = W^{\mathrm{T}}D\boldsymbol{y}$,

making it suitable for efficient execution on GPUs.

Algorithm 1 outlines the construction of W, which is based on the existing approach in [1]. Here, the (i, j) entry of A is denoted by a_{ij}, and $L' = D + \omega L$. In this algorithm, we additionally drop small entries based on a threshold τ (Line 9) to further reduce the cost of SpMV. It is also worth noting that the i-loop (Line 1) can be parallelized, thereby reducing the construction time of W.

4 Numerical experiments

We compare the proposed SSOR variant with the following existing parallel SSOR variants:

- **MC**: Forward and backward substitutions are parallelized after reordering A using a multi-coloring technique [6].

Algorithm 1 The construction of W

for $i = 1, 2, \ldots, n$ **do**
  $w_{ii} \leftarrow 1/l'_{ii}$
  for $k = i+1, \ldots, n \wedge l'_{ki} \neq 0$ **do**
    $w_{ki} \leftarrow 0$
    for $j = i, \ldots, k-1 \wedge l'_{kj} \neq 0$ **do**
      $w_{ki} -= l'_{kj} w_{ji}$
    end for
    $w_{ki} \leftarrow w_{ki}/l'_{kk}$
    if $|w_{ki}| < \tau$ **then** $w_{ki} \leftarrow 0$
  end for
end for

- **TNS**: $(D + \omega L)^{-1}$ is approximated by a p-term truncated Neumann series, enabling SpMV-based calculations [2, 3].

Experimental settings are as follows:

- **Platform:** Grand Chariot 2 at Hokkaido University; one NVIDIA H100 GPU and one Intel Xeon Gold 6548Y+ CPU.
- **Implementation:** CUDA 12.8.0; `nvcc -O3 -Xcompiler -fopenmp`; FP64 arithmetic.
- **Execution:** Setup on the CPU with 16 threads; solve phase on the GPU.
- **Parameters:** $\omega = 0.8$, $\tau = 0.01$.
- **Stopping criterion:** $\|\boldsymbol{r}_i\|/\|\boldsymbol{b}\| \leq 10^{-10}$ or 100,000 iterations.
- **Initial vectors:** $\boldsymbol{x}_0 = \boldsymbol{0}$, $\boldsymbol{b} = \boldsymbol{1}$.
- **Matrices:** The six largest SPD matrices with $n \geq 10^6$ from the SuiteSparse Matrix Collection, diagonally scaled so that $D = I$ and stored in CRS format.

Figure 1 summarizes the experimental results:

(a) **Time to compute z**: The proposed variant is faster than MC, demonstrating the benefit of replacing forward and backward substitutions with SpMVs. It is also faster than TNS because W has fewer nonzero entries.

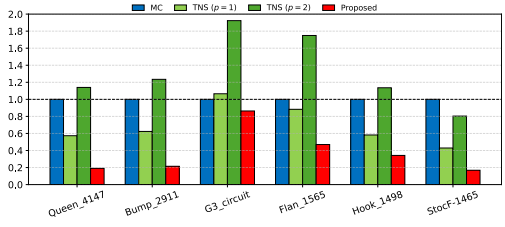
(b) **Iteration count:** The proposed variant improves the convergence of CG to a similar extent as MC and more than TNS for some matrices.

(c) **Total time of PCG (setup and solve):** The proposed variant is faster than MC and TNS in all but one case, despite the additional cost of constructing W.

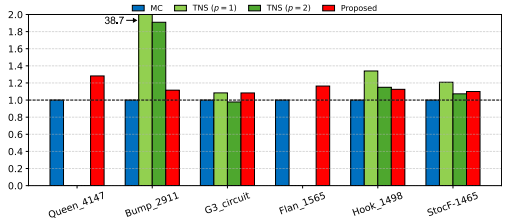
Overall, the proposed variant substantially reduces the cost of applying the preconditioner by replacing the forward and backward substitutions with SpMVs, while largely preserving the convergence improvement of SSOR. Consequently, despite the additional cost of constructing W, the total PCG execution time is reduced for most of the test matrices, demonstrating that the proposed approach provides an effective GPU-friendly SSOR variant.

5 Conclusion

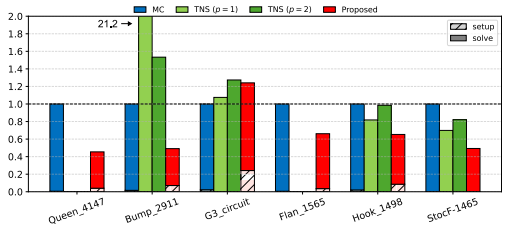
We developed a GPU-friendly variant of the SSOR preconditioner based on an approximate inverse of a triangular matrix and demonstrated its effectiveness on an NVIDIA H100 GPU. Future work includes optimizing the parameters ω and τ, comparing the proposed SSOR variant with other preconditioners, such as IC and AINV, and exploiting mixed-precision computing. We also plan to extend the method to nonsymmetric linear systems.



(a) Time to compute z



(b) Iteration count



(c) Total time (setup and solve)

Figure 1: Comparison of the proposed and existing parallel SSOR variants. All values in each graph are normalized by the corresponding values for MC. Smaller values are better.

Acknowledgments

This work was supported by JST SPRING Grant Number JPMJSP2119 and JSPS KAKENHI Grant Numbers JP25K21213, JP25H01109, and JP26H02492. This research was also supported by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" and "High Performance Computing Infrastructure" in Japan (Project ID: jh260083 and jh260024).

References

[1] Hartwig Anzt, Thomas K. Huckle, Jürgen Bräckle, and Jack Dongarra. 2018. Incomplete Sparse Approximate Inverses for Parallel Preconditioning. *Parallel Comput.* 71 (2018), 1–22. doi:10.1016/j.parco.2017.10.003

[2] Michele Benzi and Miroslav Tůma. 1999. A comparative study of sparse approximate inverse preconditioners. *Appl. Numer. Math.* 30, 2 (1999), 305–340. doi:10.1016/S0168-9274(98)00118-4

[3] Rudi Helfenstein and Jonas Koko. 2012. Parallel preconditioned conjugate gradient algorithm on GPU. *J. Comput. Appl. Math.* 236, 15 (2012), 3584–3590. doi:10.1016/j.cam.2011.04.025

[4] Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems* (2nd ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA. doi:10.1137/1.9780898718003

[5] Kengo Suzuki, Takeshi Fukaya, and Takeshi Iwashita. 2022. A New AINV Preconditioner for the CG Method in Hybrid CPU-GPU Computing Environment. *J. Inf. Process.* 30 (10 2022), 755–765. doi:10.2197/ipsjjip.30.755

[6] Changhui Wu. 1990. A multicolour SOR method for the finite-element method. *J. Comput. Appl. Math.* 30, 3 (1990), 283–294. doi:10.1016/0377-0427(90)90280-D