Parallel Incomplete Factorization Preconditioners

Erik G Boman[1], *Marc A. Tunnell*[2]

Abstract

Incomplete factorizations are popular preconditioners and are well known to be effective for a wide range of problems. Additionally, these preconditioners can be used as a "black box" and do not rely on any *a priori* knowledge of the problem. However, traditional algorithms for computing these incomplete factorizations are based on Gaussian elimination and do not parallelize well. Recently, a more parallel incomplete factorization algorithm was proposed by Chow and Patel [4], where the factors are computed iteratively. Here we propose a new iterative approach that is based on alternating triangular solves of L and U. We develop two versions: ATS-ILU for a static sparsity pattern, and ATS-ILUT for a dynamic pattern (using thresholding). We show that this new method is similar to the fine-grained iterative ILU method by Chow but has the added advantage that it allows greater reuse of memory and is fully deterministic in parallel, meaning the results do not depend on scheduling. We evaluate the new method on several test matrices from the SuiteSparse collection and show that it is competitive with current ILU methods. When short setup time is important, it is typically better than other methods.

1 Introduction

Preconditioning is well known to be essential for improving the speed of convergence of Krylov methods such as Conjugate Gradient (CG) and Generalized Minimal Residual (GMRES) [8]. Incomplete Lower-Upper (ILU) factorizations are a popular class of preconditioners as they can be used as a "black box" on a wide range of problems. There are two main types of ILU factorizations, level-based ILU(k) [3, 6, 7] and threshold-based ILUT [9]. However, these methods are inherently sequential and do not parallelize well.

There has been interest in the parallelization of these more classical interpretations of ILU, largely through graph partitioning schemes. These graph partition-based methods, such as [5], offer a promising approach to parallelizing classical ILU methods. By decomposing the graph corresponding to the matrix and determining variables that can be eliminated in parallel, these methods aim to distribute the computational load more evenly across processors. While these strategies have shown effectiveness for certain types of problems [3], their implementation can be highly complex. Additionally, their performance can be problem-dependent, requiring consideration of the underlying graph structure when choosing a parallelization strategy.

More recently, there have been strides into methods of computing ILU factors iteratively, potentially giving up some of the robustness of the classical methods for better parallel properties [4]. Iterative ILU methods, such as those introduced by Chow [4], offer significant advantages in terms of scalability on modern parallel architectures. For the remainder of this paper, we refer to the method introduced by Chow as ParILU and its thresholded counterpart as ParILUT [1, 4]. These methods approximate the ILU factors through a series of iterative updates, which can be more easily distributed across multiple processors or offloaded to accelerators.

[1]Sandia National Labs, egboman@sandia.gov
[2]Purdue University, mtunnell@purdue.edu

By breaking down each iterative update into smaller approximate subproblems and solving them independently, different parts of the factorization can be computed in parallel without the need for complex graph-partitioning algorithms. This approach allows for the use of iterative ILU methods on a wide range of problems, including those with complex or irregular graph structures that may preclude high levels of parallelism in the graph-partitioned classical ILU methods.

Furthermore, iterative ILU methods are adaptable to various hardware accelerators such as graphics processing units (GPUs) [2], which are increasingly important for high-performance computing. By leveraging the parallel processing capability of these accelerators, iterative ILU methods can significantly reduce the real-world time required to compute the ILU factors for large-scale problems, thereby speeding up the overall solution process.

In this paper, we propose a new class of iteratively-computed ILU preconditioners, which we call Alternating triangular Solves ILU (ATS-ILU). This method builds upon the strengths of existing iterative ILU approaches while leveraging improved memory reuse and determinism in parallel. We provide an analysis of the method and evaluate its performance compared to the state of the art on a variety of test matrices. We show that our method is competitive with current ILU methods and has the potential to be a powerful tool for solving large-scale problems on modern parallel architectures.

2 Alternating Triangular Solves Method

In this section, we introduce our new method for computing ILU factors, ATS-ILU. This method is based on the idea of alternating iterative updates to the L and U factors of the matrix A. The basic idea is the same as before, where we iteratively update the factors L and U until convergence, but where the updates are performed in an alternating manner. This general process is a common method for solving bilinear systems and is outlined in 1.

Algorithm 1 Alternating ILU

$U^{(0)} \leftarrow \texttt{triu}(A)$
$k \leftarrow 0$
while not converged **do**
  Solve $L^{(k)}U^{(k)} \approx A$ for $L^{(k)}$
  Solve $L^{(k)}U^{(k+1)} \approx A$ for $U^{(k+1)}$
  Check convergence
  $k \leftarrow k + 1$
end while

One way to perform this procedure would be to perform a triangular solve with the entirety of $U^{(k)}$ and let A be the right-hand side vector to solve for $L^{(k+1)}$, and similar to solve for $U^{(k+1)}$. This entire process can largely be performed in parallel as each row of L and column of U can be solved independently. Despite the potential for high levels of parallelism, it is still extremely computationally expensive and likely suffers from significant levels of fill-in during intermediate steps. The computational cost could be reduced by using an approximation.

Additionally, the algorithm as stated above does not guarantee that L and U remain lower and upper triangular, respectively. One method to address this issue would be to solve for L only in the lower triangular part of A and for U only in the upper triangular part of A. This would ensure that the factors remain lower and upper triangular, respectively, but would still leave the problem

of significant levels of fill-in. Instead, we suggest a more practical approach where we impose a sparsity pattern on L and U, namely $\mathcal{L}$ and $\mathcal{U}$, respectively. This sparsity pattern can be chosen to be the same as the sparsity pattern of A, which is the choice we make in this paper.

In order to get around the issue of fill-in, we propose a method where we approximately solve for L and U along their given sparsity patterns, which we discuss next.

2.1 Alternating Triangular Solves ILU Algorithm

The ATS-ILU algorithm is based on the idea of approximately solving for L and U in an alternating fashion along only their given sparsity patterns. Again, the rows of L and the columns of U can be solved independently, allowing for a high level of parallelism. The algorithm is shown in 2. We present the algorithm for a general pattern $\mathcal{S}$ but in practice, this will correspond to the pattern of A^k for some small power k.

Algorithm 2 ATS-ILU

Input: Sparse matrix A, sparsity pattern $\mathcal{S}$, starting factors L and U
while not converged **do**
  for $i \in \{1\ 2\ \ldots\ n\}$ **do**
    $\text{idx} \leftarrow \{j \in \mathbb{N} \mid (i,j) \in \mathcal{S}, j \leq i\}$
    $\ell_{i,\text{idx}} \leftarrow a_{i,idx} \left(U_{\text{idx,idx}}\right)^{-1}$
  end for
  for $j \in \{1\ 2\ \ldots\ n\}$ **do**
    $\text{idx} \leftarrow \{i \in \mathbb{N} \mid (i,j) \in \mathcal{S}, i \geq j\}$
    $u_{\text{idx},j} \leftarrow \left(L_{\text{idx,idx}}\right)^{-1} a_{\text{idx},j}$
  end for
end while

In this algorithm, we solve for each row of L and each column of U independently. Recall that the notation $\mathbf{a}_{i,idx}$ refers to the i^{th} row of $\boldsymbol{A}$ restricted to the indices in idx, and similarly for $U_{\text{idx,idx}}$ and $L_{\text{idx,idx}}$. These submatrices can be viewed as the (dense) non-contiguous submatrices of L and U that correspond to the sparsity pattern $\mathcal{S}$ along the given row or column.

Our method can be extended to do thresholding to maintain a certain fill level. We call this extension ATS-ILUT, and defer the details to the full paper.

3 Results

We implemented the ATS-ILUT algorithm in C++ with Kokkos for parallel performance portability. We show some preliminary results in Table 3.

4 Conclusions

We have developed a new parallel iterative ILU algorithm ATS-ILU and a thresholded version ATS-ILUT. Experiments show it performs similarly to the ParILU(T) method, but it often provides a better quality preconditioner after just one or two steps (updates) of the setup. This is an advantage

Table 1: Comparison of ATS-ILU Variants with PAR-ILUT across Different Matrices, Fill Levels, and Iterations. The best at each iteration is bolded.

Matrices:		abnormal_sandia					af_shell3					G3_circuit					parabolic_fem				
Method	Fill	Iterations																			
		1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
ATS-ILUT (a)	1.0	54	**44**	**42**	**42**	**42**	905	723	594	**605**	**568**	1169	**1140**	**1148**	**1133**	**1131**	**1183**	**1089**	**1090**	**1083**	**1071**
	2.0	51	33	**25**	**24**	**23**	872	564	395	334	299	860	639	467	426	396	765	561	464	492	444
	3.0	51	31	22	**18**	**17**	872	559	378	308	258	860	638	448	373	318	765	551	422	434	381
ATS-ILUT (b)	1.0	**50**	45	45	46	45	**797**	657	638	625	631	**1153**	1186	1189	1187	1183	1261	1192	1224	1220	1201
	2.0	**42**	**29**	27	27	27	**651**	**402**	319	289	274	**690**	**520**	408	424	414	**729**	719	505	547	446
	3.0	**42**	**24**	**20**	20	20	**651**	**397**	**290**	**226**	**204**	**690**	**520**	**357**	326	305	**729**	715	681	525	378
ParILUT	1.0	54	45	45	45	45	822	**597**	**581**	616	592	1188	1170	1180	1215	1217	1232	1168	1190	1201	1197
	2.0	49	32	26	25	25	752	415	**311**	**279**	**268**	758	531	**390**	**365**	**360**	864	**482**	**379**	**353**	**354**
	3.0	49	30	21	**18**	**17**	752	415	293	234	**204**	758	531	379	**295**	**269**	864	**479**	**320**	**219**	**191**

if setup time is important, e.g., when solving a sequence of linear systems. Also, it is naturally deterministic (though an asynchronous version is also possible).

Acknowledgments

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.

References

[1] Hartwig Anzt, Edmond Chow, and Jack Dongarra. Parilut—a new parallel threshold ILU factorization. *SIAM Journal on Scientific Computing*, 40(4):C503–C519, 2018.

[2] Hartwig Anzt, Tobias Ribizel, Goran Flegar, Edmond Chow, and Jack Dongarra. Parilut - a parallel threshold ILU for GPUs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, May 2019.

[3] Michele Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, November 2002.

[4] Edmond Chow and Aftab Patel. "A fine-grained parallel ILU factorization". *SIAM Journal on Scientific Computing*, 37(2):C169–C197, 2015.

[5] David Hysom and Alex Pothen. Efficient parallel computation of ILU(k) preconditioners. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, SC '99. ACM, January 1999.

[6] Na Li, Yousef Saad, and Edmond Chow. Crout versions of ilu for general sparse matrices. *SIAM Journal on Scientific Computing*, 25(2):716–728, January 2003.

[7] Y Saad. *Iterative methods for sparse linear systems*. SIAM, Philadelphia, MS, 2 edition, 2003.

[8] Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986.

[9] Yousef Saad. ILUT: A dual threshold incomplete lu factorization. *Numerical Linear Algebra with Applications*, 1(4):387–402, July 1994.