Brief Announcement: Communication Optimal Sparse LU Factorization for Planar Matrices

Piyush Sao*
Oak Ridge National Laboratory
Oak Ridge, TN
saopk@ornl.gov

Xiaoye Sherry Li
Lawrence Berkeley National Laboratory
Berkeley, CA
xsli@lbl.gov

ABSTRACT

We introduce a new parallel algorithm for solving sparse LU factorization of planar matrices, which commonly arise in the finite element method for 2D PDEs. Existing scalable methods, such as the multifrontal approach with *subtree-to-subcube mapping* by Gupta et al. [1] and right-looking with *3D mapping* by Sao et al. [2], fail to achieve optimal communication costs for these matrices. Our new algorithm combines 3D mapping and subtree-to-subcube mapping to minimize communication costs while allowing trade-offs between extra memory and reduced communication. We demonstrate that our proposed algorithm attains the communication lower bound up to a factor of $O(\log\log n)$ in the memory-optimal case and up to a factor of $O(\log P)$ in the memory-independent case for an n-dimensional planar sparse matrix on P processors.

CCS CONCEPTS

• **Theory of computation** → **Parallel algorithms**; • **Mathematics of computing** → **Solvers**; *Numerical analysis*;

KEYWORDS

Sparse LU factorization, communication optimal algorithms

ACM Reference Format:
Piyush Sao and Xiaoye Sherry Li. 2023. Brief Announcement: Communication Optimal Sparse LU Factorization for Planar Matrices. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '23), June 17–19, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3558481.3591315

This work was supported by the U.S. Department of Energy (DOE), Office of Science, Advanced Scientific Computing Research (ASCR) through the Mathematical Multifaceted Integrated Capability Centers (MMICCS) program, Program Manager William Spotz, under Award No. ERKJ401.

SPAA '23, June 17–19, 2023, Orlando, FL, USA

1 INTRODUCTION

Our goal is to understand the minimum data transfer required between different processors to perform LU factorization of planar sparse matrices—matrices associated with planar graphs. We often encounter such matrices in the finite element method for solving two-dimensional PDEs, making them crucial in engineering and science. In 2004, Irony and Toledo [3] established the communication limits for distributed memory matrix multiplication as:

$$W = \Omega\left(F/\sqrt{M}\right), \tag{1}$$

where W represents per-process communication volume, F indicates per-process floating-point operations performed, and M denotes per-process memory. Researchers have developed algorithms that achieve these limits for dense matrix operations, including LU and QR factorization [4]. However, its unclear whether Equation (1) applies to general sparse computations. For instance, the sparse LU factorization algorithm can achieve these limits for sparse matrices with three or higher dimensions [5]. Yet, no known parallel sparse LU factorization algorithm meets this lower bound for planar problems. To address this gap, we propose a novel algorithm that achieves the communication-lower bounds for sparse LU factorization of planar sparse matrices.

Two scalable approaches for sparse LU factorization exists: the multifrontal approach with subtree-to-subcube mapping proposed by Gupta et al. [1], and the right-looking 3D mapping approach proposed by Sao, Vuduc and Li [2]. Both methods employ the elimination-tree structure for computation mapping but differ in mapping subtrees to smaller grids. For sparse matrices with three and higher dimensions, both approaches are communication-optimal. However, neither method reaches the lower bound for communication Equation (1) in the case of planar sparse matrices. Specifically, for a planar model problem derived from the finite difference method on a 2D grid of size $n = m \times m$, the subtree-to-subcube and 3D approaches fall short by factors of $\sqrt{\log n}$ and $\log n$, respectively. Although the 3D approach can reduce communication complexity by using extra memory, compared to the multifrontal approach, it still fails to meet memory-dependent lower bounds of Equation (1).

We propose a hybrid algorithm that combines the strengths of the 3D mapping and subtree-to-subcube approaches. Our key insight is that replicating data up to a factor of $\log n$ for higher-level subtrees does not increase the overall memory cost, and enables us to optimize communication. Specifically, we use 3D mapping to factor higher-level subtrees, taking advantage of the memory-communication trade-off, and apply subtree-to-subcube mapping to lower levels of the elimination tree. Our algorithm achieves the communication lower bound within a factor of $\log\log n$ without increasing asymptotic memory cost, outperforming existing methods.

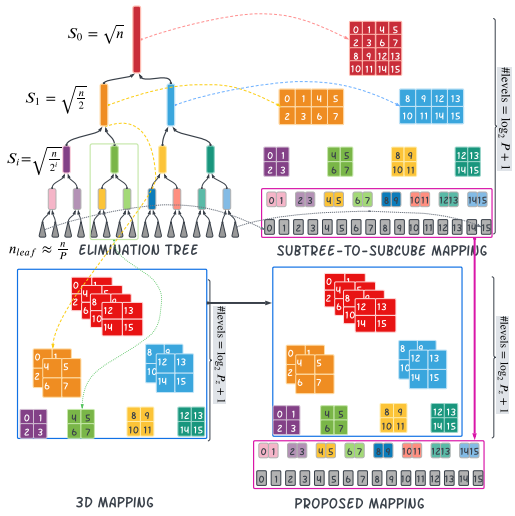

Figure 1: Different subtree to process mapping strategy for the sparse LU factorization

Our algorithm achieves an optimal trade-off between memory and communication. Specifically, it reduces communication volume by a factor of $\sqrt{c}/\log c$ while increasing memory usage by a factor of c, where c is a tunable parameter. Additionally, the algorithm attains the communication lower bound, unconstrained by memory, up to a factor of $\log P$.

2 BACKGROUND

Sparse LU factorization: Sparse LU factorization solves the equation $Ax = b$ by factoring matrix A into lower and upper triangular matrices L and U. We then use forward and back substitution to compute the solution $x = U^{-1}L^{-1}b$. The L and U factors are denser than matrix A due to non-zeros introduced during factorization. We use a fill-in-reducing ordering to minimize fill-ins during factorization, as they depend on variable elimination order.

Nested dissection reordering: Nested dissection (ND) is an efficient graph partitioning method for ordering sparse matrices. Given the graph $G = (V, E)$ associated with the sparse matrix A, ND finds a vertex separator S that divides V into $C_1 \cup S \cup C_2$, where C_1 and C_2 have no edges between them. Factoring C_1 and C_2 creates new non-zeros only in S. The matrix is then permuted, indexing vertices in S after those in C_1 and C_2, which allows the elimination of S following C_1 and C_2. This dependency forms an elimination tree (etree), and the process recursively continues for C_1 and C_2, forming a multi-level etree as shown in Figure 1. During factorization, separator blocks become dense due to fill-ins. The computation and memory needed to compute and store the L and U factors depend on the size of the largest separator block.

Sparse LU factorization of planar sparse matrices: We consider a two-dimensional grid with dimensions $n = m \times m$ as our model problem for the paper. The top-level separator for this graph has a dimension of $m = \sqrt{n}$. In general, for any planar graph with n vertices, one can find a balanced separator of size $O(\sqrt{n})$ due to the planar-separator theorem[6].

Computational cost and required memory: The total number of floating-point operations (asymptotically) in the LU factorization is the same as the cost of factoring the top-level separator $F = O(m^3) = O(n^{3/2})$. At the i-th level of the etree, we have 2^i separators, each with a dimension of approximately $\sqrt{n/2^i}$. Therefore, storing LU factors at the i-th level requires memory for $2^i\sqrt{n/2^i} \times \sqrt{n/2^i} = n$ floats. With approximately $\log n$ levels in the etree, the total memory needed for storing the L and U factors is approximately $n \log n$ floats.

Multifrontal with subtree-to-subcube mapping: We show the multifrontal method, which uses a subcube-to-subtree mapping, in Figure 1. This method divides the elimination tree into subtrees and maps them to smaller 2D processor grids. At the top level of the elimination tree, we solve a dense problem with size $m = \sqrt{n}$ using a 2D grid of $\sqrt{P} \times \sqrt{P}$ processors, where P is the total processor count. As we move down the elimination tree, the subtrees and process grid dimensions become smaller while the number of subtrees increases. At level i, we have 2^i separators, each with dimension $O(\sqrt{n/2^i})$. We solve each separator in parallel on a 2D grid with dimensions $\sqrt{P/2^i} \times \sqrt{P/2^i}$. This process continues until we reach $logP$ levels. In the final level, each of the P processors solves a problem of size $\approx n/P$. As we descend the etree, communication decreases for each processor. The top level has $n/\sqrt{P}$ per-process communication, while level i has $n/\sqrt{2^iP}$. The total per-process communication, $W_{mf}(n, P)$, is given by the following equation:

$$W_{mf}(n, P) = O\left(n/\sqrt{P}\right) \tag{2}$$

Right-looking 3D sparse LU factorization: In the 2D distributed right-looking LU factorization, where we distribute all the LU factors using a block-cyclic manner. This method balances load using 2D block-cyclic distribution, making it suitable for graphs with arbitrary shapes. However, for planar cases, it requires $W = O\left(n \log n/\sqrt{P}\right)$ asymptotic communication, which is not optimal.

The 3D mapping: To address the limitations of the 2D approach, we proposed a 3D algorithm that employs a 3D process grid, as illustrated in Figure 1. We arrange P processes into P_z 2D process grids, each with dimension $P_{xy} = P_x \times P_y$, such that $P = P_{xy} \times P_z$. In the lowest level, we factorize problems of size n/P_z in each of the P_z 2D grids. As we ascend the tree, the 3D algorithm merges 2D grids into 3D grids rather than extending a 2D grid to a larger 2D grid as in the subtree-to-subtree mapping. This approach allows us to factorize higher-level separators using a 2.5D factorization algorithm, reducing the communication costs of right-looking LU factorization by a factor of $\sqrt{P_z}$, resulting in

$$W_{3d} = O\left(n \log n/\sqrt{PP_z}\right) \tag{3}$$

However, this approach increases memory requirements by replicating data in each 2D grid, leading to

$$M_{3d} = M_{2d}O\left(1 + P_z/\log n\right) \tag{4}$$

When $P_z = O(\log n)$, the asymptotic memory requirement remains unchanged. In this scenario, the per-process communication cost is $W_{3d} = n\sqrt{\log n}/\sqrt{P}$, which, although improved, is still not optimal.

Communication lower bounds of LU factorization of planar matrix: For the model problem of size n on P processes, the per-process floating operations F equal $O\left(n^{3/2}/P\right)$, and the per-process

memory (M) equals $O(n \log n/P)$. The communication lower bounds for this computation using Equation (1) are given by:

$$W_\Omega(n,P) = \Omega\left(n/\sqrt{P \log n}\right). \tag{5}$$

It is worth noting that the 3D algorithm can achieve $W_{3d} = W_\Omega$ by setting $P_z = \log^3 n$. However, this adjustment increases the asymptotic memory to $M = n \log^3 n/P$ and shifts the lower bound of Equation (1) to $W_\Omega/\log n$. As a result, the 3D algorithm remains off by a factor of $\log n$ from the memory-dependent communication lower bound.

3 A NEW "3D" SPARSE LU ALGORITHM

In this section, we introduce our new 3D algorithm that combines 3D mapping with subtree-to-subcube mapping techniques to achieve communication-optimality for planar sparse matrices. Our algorithm leverages a parameter $P_z = 2^{l_z}$ to divide the process of factorization into two parts. In the first part, we employ 3D mapping to factor levels 0 to $l_z - 1$ of the elimination tree. In contrast, the second part uses subtree-to-subcube mapping to factor levels l_z to $\log P$ of the elimination tree. We show this partition in Figure 1. Our goal is to demonstrate that this novel algorithm achieves a communication lower bound within a $\log P_z$ factor. To compute the total communication cost, we consider the communication in two parts of factorization: the 3D part, denoted as W_{anc}, and the subtree-to-subcube part, denoted as W_{leaf}. The total communication is the sum of these two components: $W_{anc} + W_{leaf}$.

Computing W_{leaf}: We factor P_z-sparse matrices of size $\approx n/P_z$ in parallel with using P/P_z processes using subtree to subcube mapping. Hence, using Equation (2), W_{leaf} is given by:

$$W_{leaf}(n,P) = W_{mf}(n/P_z, P/P_z) = O\left(n/\sqrt{PP_z}\right) \tag{6}$$

Computing W_{anc}: First, recall that we factor a dense LU matrix of dimension m using the "2.5D" algorithm by Solomonik and Demmel [7] with P processes. The per-process communication cost is given by:

$$W_{2.5D}(m,P,P_z) = O\left(m^2/\sqrt{PP_z} + m^2P_z/P\right) \tag{7}$$

When $P_z \le P^{1/3}$, we consider only the first term in Equation (8):

$$W_{2.5D}(m,P,P_z) = O\left(m^2/\sqrt{PP_z}\right) \text{ for } P_z \le P^{1/3} \tag{8}$$

Using Equation (8), we calculate and sum the per-process communication in each level $i = 0{:}l_z - 1$ to obtain W_{anc}. In level i, we perform factorization on 2^i dense matrices in parallel, using 2^i groups of processors. Each matrix in level i has a dimension $m_i = \sqrt{n/2^i}$, and we use a total of $P^i = P/2^i$ processes arranged in a 3D grid with $P_z^i = P_z/2^i$. The per-process communication in level i is given by Equation (8) as:

$$W_{anc}^i = m_i^2/\sqrt{P^iP_z^i} = O\left(n/\sqrt{PP_z}\right).$$

By summing the per-process communication for all levels, we obtain:

$$W_{anc}(n,P) = \sum_{i=0}^{l_z-1} W_{anc}^i = nl_z/\sqrt{PP_z}$$

Finally, we determine the total communication cost, W_{new}, as the sum of W_{anc} and W_{leaf}:

$$W_{new} = W_{anc} + W_{leaf} = \frac{n}{\sqrt{PP_z}}(1 + \log P_z) = O\left(\frac{n \log P_z}{\sqrt{PP_z}}\right) \tag{9}$$

Per-process memory: The new algorithm's per-process memory cost is the same as that of the 3D algorithm. Therefore, the per-process memory cost is given by $M_{\text{new}} = M_{3d}$.

Analysis of Communication Cost: We analyze the communication cost of the new 3D algorithm in three cases: the memory-optimal case, the constrained memory case, and the case with no memory constraints. For each case, we discuss how to select the best P_z.

Memory optimal case $M = O(M_{2D})$: If we want to maintain the asymptotic memory requirement per process, we can only set $P_z = O(\log n)$. In this case, we calculate the communication cost of the new 3D algorithm as:

$$W_{new}(n,P)_{|\ M=M_{2D}} = O\left(\frac{n}{\sqrt{P \log n}} \log\log n\right) \tag{10}$$

Hence, we conclude that $W_{new}(n,P)_{|\ M=M_{2D}} = W_\Omega(n,P)O(\log\log n)$, where $W_\Omega(n,P)$ is the theoretical communication lower derived in Equation (5).Hence, the new 3D algorithm achieves communication optimality within a factor of $O(\log\log n) = O(\log P_z)$.

Trading extra memory for reduced communication $M = cM_{2D}$ and $1 < c \le P^{1/3}/\log n$: Using extra memory allows us to decrease communication time by selecting a P_z value larger than $\log n$. When we set an allowed memory level as $M = cM_{2D}$, where c is no greater than $P^{1/3}/\log n$, the optimal P_z becomes $c \log n$. Consequently, the communication time is given by

$$W_{new}(n,P)_{|\ M=cM_{2D}} = O\left(\frac{n}{\sqrt{cP \log n}} (\log c + \log\log n)\right) \tag{11}$$

as shown in equation (11). The limitation of $c \le P^{1/3}/\log n$ results from the condition $P_z \le P^{1/3}$.

Memory independent communication bounds: Indefinitely increasing P_z is not viable for reducing communication costs. This is because the 2.5D LU algorithm performs optimally when $P_z \le P^{1/3}$. When P_z exceeds this threshold, i.e., $P_z > P^{1/3}$, the m^2P_z/P term in Equation (7) becomes dominant. Therefore, in scenarios without memory constraints, we choose $P_z = P^{1/3}$, resulting in a memory-independent communication cost given by

$$W_{new}(n,P)_{|\ P_z=P^{1/3}} = O\left(n \log P/P^{2/3}\right). \tag{12}$$

Thus, in all cases, our proposed 3D algorithm achieves communication costs within $O(\log P)$ factor of optimal for planar sparse matrices.

4 CONCLUSION

We developed a new sparse LU factorization algorithm for planar matrices that achieves communication optimality by combining in 3D and subtree-to-subcube mapping. In general, establishing achievable communication lower bounds for sparse operations can be difficult, as they depend on the sparsity pattern of the matrix. By applying similar strategies to related issues, such as sparse triangular solvers [8] and all-pair shortest path algorithms [9], we may potentially uncover new insights and develop novel algorithms.

REFERENCES

[1] A. Gupta, G. Karypis, and V. Kumar, "Highly scalable parallel algorithms for sparse matrix factorization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 5, pp. 502–520, 1997.

[2] P. Sao, X. S. Li, and R. Vuduc, "A communication-avoiding 3D LU factorization algorithm for sparse matrices," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, (Vancouver, BC, Canada), May 2018.

[3] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1017–1026, 2004.

[4] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential qr and lu factorizations," *SIAM Journal on Scientific Computing*, vol. 34, no. 1, pp. A206–A239, 2012.

[5] L. Grigori, P.-Y. David, J. W. Demmel, and S. Peyronnet, "Brief announcement: Lower bounds on communication for sparse cholesky factorization of a model problem," in *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*, pp. 79–81, ACM, 2010.

[6] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM Journal on Applied Mathematics*, vol. 36, no. 2, pp. 177–189, 1979.

[7] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms," in *Euro-Par 2011 Parallel Processing*, pp. 90–109, Springer, 2011.

[8] P. Sao, R. Kannan, X. S. Li, and R. Vuduc, "A communication-avoiding 3d sparse triangular solver," in *Proceedings of the ACM International Conference on Supercomputing*, pp. 127–137, 2019.

[9] P. Sao, R. Kannan, P. Gera, and R. Vuduc, "A supernodal all-pairs shortest path algorithm," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 250–261, 2020.