A communication-avoiding 3D sparse triangular solver

Piyush Sao, Ramakrishnan Kannan[*]
Oak Ridge National Laboratory
Oak Ridge, TN
{saopk,kannanr}@ornl.gov

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

Richard Vuduc[‡]
Georgia Institute of technology
Atlanta, GA
richie@gatech.edu

ABSTRACT

We present a novel distributed memory algorithm to improve the strong scalability of the solution of a sparse triangular system. This operation appears in the solve phase of direct methods for solving general sparse linear systems, $Ax = b$. Our 3D sparse triangular solver employs several techniques, including a 3D MPI process grid, elimination tree parallelism, and data replication, all of which reduce the per-process communication when combined. We present analytical models to understand the communication cost of our algorithm and show that our 3D sparse triangular solver can reduce the per-process communication volume asymptotically by a factor of $O\left(n^{1/4}\right)$ and $O\left(n^{1/6}\right)$ for problems arising from the finite element discretizations of 2D "planar" and 3D "non-planar" PDEs, respectively. We implement our algorithm for use in SuperLU_DIST3D, using a hybrid MPI+OpenMP programming model. Our 3D triangular solve algorithm, when run on 12k cores of Cray XC30, outperforms the current state-of-the-art 2D algorithm by 7.2x for planar and 2.7x for the non-planar sparse matrices, respectively.

KEYWORDS

sparse matrix computations, distributed-memory parallelism, communication-avoiding algorithms

ACM Reference Format:
Piyush Sao, Ramakrishnan Kannan, Xiaoye Sherry Li, and Richard Vuduc. 2019. A communication-avoiding 3D sparse triangular solver. In *2019 International Conference on Supercomputing (ICS '19), June 26–28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3330345.3330357

[*]This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).
[†]Supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy (DOE) Office of Science and the National Nuclear Security Administration (NNSA).
[‡]Supported in part by the Defense Advanced Research Projects Agency (DARPA) contract FA8750-18-2-0108, under the DARPA MTO Software Defined Hardware program, and the National Science Foundation (NSF) under Award 1710371.

ICS '19, June 26–28, 2019, Phoenix, AZ, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6079-1/19/06…$15.00
https://doi.org/10.1145/3330345.3330357

1 INTRODUCTION

This paper presents a new algorithm for solving a sparse triangular system of linear equations, $Tx = b$, where T is either an upper- or lower-triangular sparse matrix. A sparse triangular solver (SpTrs) is an important sub-step during LU and Cholesky factorization, which are direct methods for solving general linear systems. SpTrs also appears in preconditioners based on incomplete factorization, which commonly appear in Krylov subspace-based iterative methods.

In the context of distributed memory sparse direct methods for solving $Ax = b$, where A is any general matrix, consider the example of sparse LU factorization. It first decomposes A into the product $A = LU$, where L and U are lower- and upper-triangular matrices, respectively. Then, one may solve for x by a pair of SpTrs operations, $Ly = b$ and $Ux = y$. In this setting, the factorization step (determining L and U) usually dominates the pair of SpTrs operations. However, a common use-case for sparse direct solvers is using *many* right-hand sides for a fixed matrix (pattern). This scenario occurs in time-stepping numerical ODE solvers, where b changes at each time step. Similarly, in the case of a sparse iterative solver, we might factor the system once upfront and then invoke SpTrs with a new right-hand side during each iteration. Thus, the scalability of SpTrs can also become a bottleneck.

In our previous work, we developed a communication-avoiding algorithm for LU factorization [21]. The idea underlying this SuperLU_Dist3D method is to organize the MPI processes logically into a three-dimensional grid, rather than a traditional 2D one, and then exploit the structure of the *elimination tree*—an abstraction that captures the data dependencies in sparse LU factorization—to replicate data judiciously. This combination of techniques provably reduce communication asymptotically in the problem size in common cases. In this work, we leverage the 3D sparse LU data structure of SuperLU_Dist3D to develop a communication-avoiding SpTrs, which yields asymptotic reductions in the latency and communication-volume costs of a conventional SpTrs.

Briefly, our new 3D SpTrs works as follows. Consider the 3D process grid as a collection of 2D MPI process grids. The prior technique of SuperLU_Dist3D mapped independent subtrees of the elimination tree to each 2D process grid and replicated the common ancestors. Our 3D triangular solver exploits this same 3D organization. It first solves independent subtrees on different 2D process grids, and then performs a reduction before solving the subproblem in the common ancestor tree on a single 2D grid.

To analyze the communication and latency costs of our new method, we consider prototypical matrices arising from the discretization of "planar" and "non-planar" partial differential equations (PDEs). By planar, we mean the physical geometry of the input domain, when discretized, is flat or nearly so; we use the term planar instead of 2D to distinguish the problem geometry from that of the logical MPI process grid. Our analysis shows that the 3D SpTrs changes the communication and latency costs by a factor of $O\left(\frac{1}{\sqrt{p_z}}\right)$ over a purely 2D algorithm, where p_z is the number of 2D process grids. This advantage comes at the cost of a small amount of additional memory needed to replicate the right-hand side.

We present empirical scalability results for our 3D SpTrs on up to 24k cores of a Cray XC30 machine. For a single right-hand side, our 3D SpTrs achieves a 4.6× and 1.8× speedup over the baseline 2D algorithm for planar and non-planar matrices, respectively. For multiple right-hand sides, our 3D SpTrs achieves 7.2× and 2.7× speed-up over the baseline 2D algorithm for planar and non-planar matrices, respectively. While our context is triangular solves for in direct methods, without loss of generality, our methods can be extended to general cases as well. Moreover, an important consequence is that SpTrs can actually improve the direct solver itself (see Section 7).

2 BACKGROUND

To understand the new algorithm (Section 3) and its analysis (Section 4), this section starts by explaining how triangular systems arise in sparse direct solvers and summarizes a baseline parallel algorithm. It then briefly reviews our previous 3D sparse LU data structure [21], upon which our new SpTrs also depends.

Terminology. In numerical linear algebra software, a triangular solver for a single right-hand side is also known as xTrsv, and for multiple right-hand sides, xTrsm. Typically, these are optimized differently in the single-node case. However, the focus of this paper is on distributed memory scalability, where such distinctions are less important, and we use the term SpTrs to denote either case. The important distinction is between the baseline sparse triangular solver algorithm, denoted SpTrs2D, which uses a 2D process grid, and our new 3D algorithm, SpTrs3D.

2.1 Structure of a Sparse Direct Solver

A sparse direct solver for $Ax = b$ has three main steps: *preprocessing*, *numerical factorization*, and the *solve step*.

预处理过程的作用
象征分解也在该过程中

In preprocessing, the matrix A is permuted to improve the numerical stability and to reduce the *fill-ins* in L and U factors. This step also involves a *symbolic factorization*, which computes the fill-in structure and sparse meta-data for the numerical factorization.

Numerical factorization computes the unit lower triangular L and the upper triangular U factors so that $A = LU$.

The solve step calculates y for the lower triangular system $Ly = b$ for y followed by solving the upper triangular system $Ux = y$ to find the final solution x.

When there is only one right-hand side b, then numerical factorization is generally the most expensive step. Consequently, sparse data structures are "tuned" for this step, and SpTrs is designed to use that data structure. Our prior work to improve numerical factorization introduced a new 3D data structure [21], leading naturally to the new algorithm of this paper.

2.2 Triangular Systems

To better understand SpTrs, we begin with the simpler case of a dense system.

2.2.1 Dense triangular solver. A triangular system can be solved immediately due to its structure. Consider, a lower triangular matrix $Lx = b$ for solving $x_1, \cdots, x_n$, first one computes $x_1 = b_1/l_{11}$, substitute the computed x_1 into the second equation and solve for x_2. This process of solve-and-substitute is carried out sequentially until all x_i's, $\forall i \in [1, n]$ are found, as shown in Algorithm 1. When the matrix is upper triangular, the process of solve-and-substitute is carried out in reverse order, i.e., x_n is solved first and x_1 in the last, where n denotes the dimension of the system. The process of solving lower and upper triangular systems are also called forward-substitution and backward-substitution, respectively.

Algorithm 1 Forward substitution algorithm for solving lower triangular system of equation $Ly = b$

```
1: function LSolve($L$,$b$):
2:     $n \leftarrow \dim(L)$
3:     for $i = \{1,\ 2 \ldots, n\}$ do:
4:         $y_i \leftarrow \frac{1}{l_{ii}}\left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j\right)$
5:     return $y$
```

直接求解的算法

2.2.2 Triangular systems in sparse direct solvers. Triangular systems that arise from sparse direct solvers have a recursive block-arrowhead structure. Figure 1 illustrates a 3×3 block sparse matrix A, and its final L and U factors.

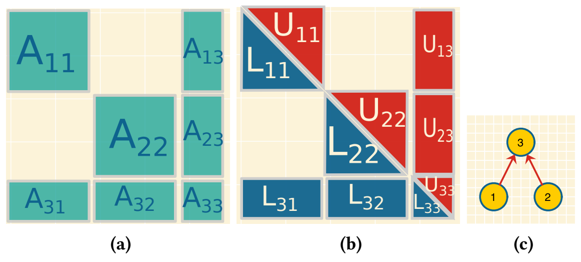
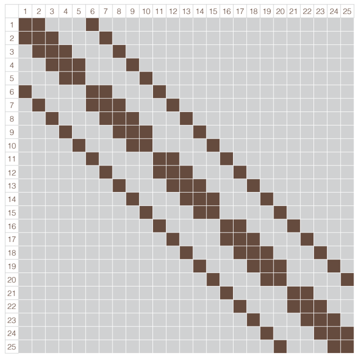
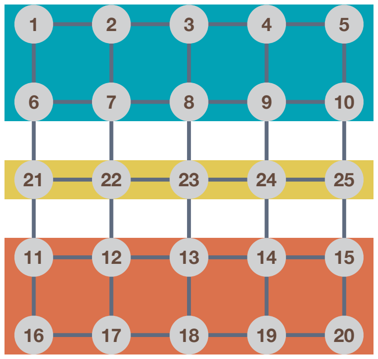


Figure 1: A 3×3 block sparse "arrowhead" matrix, its L and U factors and its block-elimination tree.

Consider a triangular system $Ly = b$, where L is the 3×3 lower triangular matrix shown in Fig. 1. The L_{21} block is zero; therefore, $L_{11}y_1 = b_1$ and $L_{22}y_2 = b_2$ can be solved concurrently. Following that, $L_{33}y_3 = b_3 - L_{31}y_1 - L_{32}y_2$ can be solved for y_3. This dependency in solution of block 3×3 lower triangular system is shown as a directed-acyclic graph (DAG) in Fig. 1. The dependency in solving L is the same as the dependency in elimination of nodes in the numerical factorization step, and the dependency DAG structure is also referred to as the *elimination* tree, or etree.


(a) A 25×25 sparse matrix



(b) The associated graph and separator

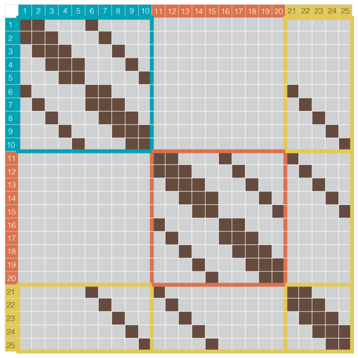

(c) Reordered sparse matrix using ND ordering

Figure 2: A sparse matrix (Fig. 2a), its associated graph (Fig. 2b), and a separator (highlighted in yellow); and the re-ordered matrix (Fig. 2c) using nested dissection (ND) ordering. The ND orders the variables so that the variables corresponding to the separator are numbered last.

2.3 Dependencies in a Sparse Triangular Solver

The block sparse matrix shown in Fig. 1 comes from the so-called nested-dissection (ND) ordering of the input matrix [6]. Such an ordering heuristically reduces nonzero fill-ins in the L and U matrices. It also exposes parallelism in sparse LU factorization and triangular solve.

Briefly, an ND ordering works as follows. Any sparse matrix A has an associated graph G, which has the same number of vertex as the dimension of A; and, for any non-zero entry a_{ij} in A, there is an edge in G from vertex v_i to v_j. For instance, in Fig. 2a, we show a 25×25 sparse matrix that arises from finite difference discretization of a 5×5 grid is shown in Fig. 2b. The ND ordering partitions in the graph G into three disjoint vertex set $\{C_1, S, C_2\}$ such there are no edges from any vertex in C_1 to any vertex in C_2. The vertex set S is called *separator*. Using this partition, we reorder A so that the vertices in S are numbered last. In Fig. 2b, we highlight the separator and in Fig. 2c, we show the reordered matrix. The Fig. 1a shows a simplified block representation of the reordered matrix Fig. 2c where A_{11}, A_{22}, and A_{33} correspond to C_1, C_2, and S respectively, with remaining submatrices representing the edges that connect these partitions. The partition C_1 and C_2 are recursively dissected to get more disjoint subgraphs till each subgraph is sufficiently small. Graph partitioning tools such as METIS [15] or PT-SCOTCH [18] can be used for calculating such a partition.

As shown in Fig. 3, an ND ordering leads to a multi-level dependency tree, also known as an elimination tree or etree. Etree describes the order of elimination in the numerical factorization process. LSOLVE has the same dependency as numerical factorization, so the etree also describes the dependency in LSOLVE.

When the input matrix A is symmetric, USOLVE follows the reverse order that of LSOLVE, i.e., LSOLVE traverses the etree in a post-order or bottom-up order, whereas USOLVE traverses the etree in top-down order. For *unsymmetric* matrices, USOLVE may traverse a slightly different tree than etree in top-down order. For simplicity, let us assume that in the unsymmetric case the etree is obtained $\boxed{\text{by applying ND on the symmetric matrix } A + A^T}$. Hence, the dependency tree for USOLVE is reverse that of LSOLVE.

为啥可以？

Table 1: List of symbols used

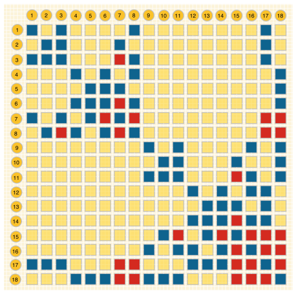
Symbol type	Symbol	Description
	P	#MPI processes
	P_x, P_y, P_z	Process grid dimensions
	P_{xy}	$P_x \times P_y$ # processes in xy plane
Process	p_x, p_y, p_z	Process coordinates
	$P_r(k)$	$(k \mod P_x)$-th process row
	$P_c(k)$	$(k \mod P_y)$-th process column
	P_{kk}	Process that owns A_{kk} block ($P_{kk} = P_r(k) \cap P_c(k)$)
	E	Elimination tree of A
	S	Top level separator of E
Graphs	$C_1,\ C_2$	Children etrees of E
	$Desc(k)$	Descendants of node k in E
	$Anc(k)$	Ancestors of node k in E
	n	Dimension of the matrix A
	l	$\log_2 P_z$
	W	Communication cost
Misc.	V	Per-process communication volume 通信量
	α	Cost of initiating a data transfer
	β	Cost of transferring a unit data
	γ	Number of right hand sides

2.4 Parallel Sparse Triangular Solve

2.4.1 SUPERLU_DIST Data Structure. Our algorithm is built on top of SUPERLU_DIST, which is an open-source sparse-direct solver library for general sparse matrices that uses right-looking scheduling and static pivoting. The baseline SUPERLU_DIST uses a two-dimensional logical process arrangement. In the two dimensional process-grid, it distributes the input matrix A into 2D block-cyclic fashion. After the factorization, A matrix is overwritten by L and U factors. Hence, L and U matrix are also distributed in a block-cyclic fashion. The right hand side b vector is distributed among the diagonal processes, so that b_k is owned by P_{kk}. Table 1 summarizes the notation used in this section.

果然分解后LU是覆盖到矩阵A上的

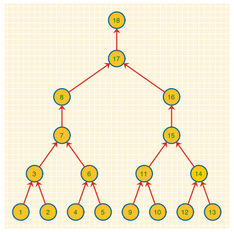
2.4.2 Distributed LSOLVE. The LSOLVE performs following operation to calculate k-th segment of solution y_k:



(a)

找分隔集
重排序到最后
重复过程

这个etree是通过多次
nested-dissection得到的



(b)

Figure 3: An 18×18 sparse matrix and its elimination tree obtained by ND ordering (Section 2.3). Here light yellow squares represent zero entries, blue entries represent non-zero entries in A, and red squares represent non-zero entries due to *fill-in* during the factorization.

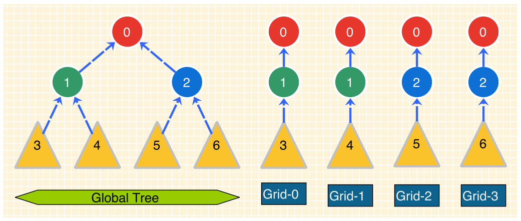
$$y_k \leftarrow L_{kk}^{-1}\left(b_k - \sum_{j \in Desc(k)} L_{kj} y_j\right). \qquad (1)$$

This operation is performed in 2D process grid using following operations. Any process $P_{kj} \in P_r(k)$, keeps a vector s_k to accumulate the local update $-L_{kj}y_j$.

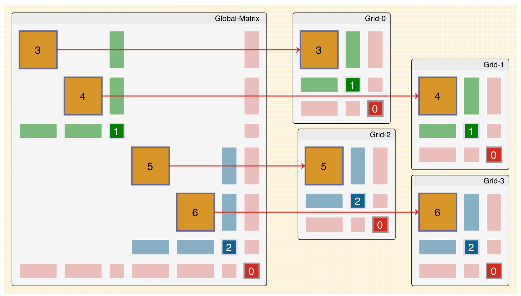
- **Local Solve:** P_{jj} solves $L_{jj}y_j = b_j$ for y_j.
- **Broadcast:** P_{jj} broadcasts the computed y_j across its process column $P_c(j)$
- **Local Update:** Any process $P_{kj} \in P_c(j)$ that owns a non-empty block L_{jk} receives y_j, and performs the local update:
  $$s_k \leftarrow s_k - L_{kj}y_j$$
- **Reduction:** When all processes in $P_r(k)$ have finished all the updates on s_k, the vector s_k is reduced across $P_r(k)$, to accumulate all the updates to P_{kk}
  $$s_k \leftarrow \sum_{i \in P_r(k)} s_k^i,$$
  where s_k^i is the s_k from the i-th process in $P_r(k)$. P_{kk} updates $b_k \leftarrow b_k - s_k$ so that
  $$b_k \leftarrow b_k - \sum_{j \in Desc(k)} L_{kj}y_j,$$
  and P_{kk} performs k-th local-solve.

In lSolve, y_k are computed in a bottom-up order of the etree.

2.4.3 Limitations of 2D lSolve. In the distributed lSolve algorithm, local-update is the main computation step, whereas broadcast and reduction are the two main communication substeps. Assuming the computation is load balanced, the local-update can exploit all the available P processors concurrently. However, each process participates in $O\left(\frac{n}{\sqrt{P}}\right)$ broadcasts and $O\left(\frac{n}{\sqrt{P}}\right)$ reductions. Therefore, the broadcast and reduction steps only scale as $1/\sqrt{P}$. Hence, the communication in lSolve does not scale as well as the computation.



(a) Etree representation of 3D data distribution



(b) Mapping of matrix blocks to 3D process grid

Figure 4: Three-dimensional data distribution in SuperLU_Dist3D [21]. In Fig. 4a we show the global elimination tree. Nodes 0 to 2 are *ancestor-subtrees* and nodes 3 to 6 are leaf subtrees. In Fig. 4b, we show how the ancestor and leaf subtrees are mapped to four 2D process grids.

2.5 3D Sparse LU factorization

Recall that our prior work developed a communication-avoiding extension of SuperLU_Dist's numerical factorization step [21]. It uses a three-dimensional data distribution instead of a 2D one. Our new algorithm SpTrS3D exploits this 3D distribution.

2.5.1 3D Data Distribution. The 3D sparse LU algorithm uses the etree to guide the data distribution for a 3D process grid. In particular, consider the 3D process grid as a collection of P_z 2D grids, where each 2D grid is of size P_{xy}. In the 3D algorithm, the etree is partitioned into independent subtrees, and each independent subtree, or *leaf subtree*, is assigned to a 2D grid. Each 2D grid also keeps a copy of the ancestors-subtree of the leaf subtree to perform the so-called *Schur-complement* update. For instance, Fig. 4a shows a two-level partition of the etree, and Fig. 4b shows how this partition is mapped to four 2D process grids. The root of the etree node-0, is replicated on all process nodes. On the other hand, node-1, and 2, are replicated on grid-0 and 1; and grid-2 and 3 respectively. In the last level, node 3 to 6 corresponds to an entire subtree of the etree, and are assigned to only one one of the 2D grid.

2.5.2 3D Factorization Algorithm. In the 3D factorization algorithm, each grid factors its leaf subtree and performs update on its copy of the ancestor subtrees. Before factoring an ancestor subtree, updates on all the copies of subtree is reduced to one process grid and then factored in 2D fashion.

At the end of the factorization, all the LU factors are gathered into a 2D grid to perform the solve step. Doing so has the following drawbacks, which this paper addresses:

- Before one can perform the solve step, all the L and U factors need to be gathered in a single 2D grid, which requires extra communication and synchronization overhead.
- The solve step can only use P_{xy} processors, and the remaining processes are idle.
- As we see in Section 4, a 2D solve algorithm has higher communication costs, thus scales poorly.

3 3D TRIANGULAR SOLVER

Our new 3D sparse triangular solver algorithm may be understood more easily by first considering a concrete example (Section 3.1, which illustrates solution of a 3×3 block sparse matrix on $P_z = 2$ 2D process grids) before presenting a more general case ($P_z = 2^l$, Section 3.2).

3.1 3×3 block sparse case

Consider the 3×3 block sparse L and U matrix distributed over two 2D process grids as shown in Fig. 1. Sparse block matrices L_{11}, L_{31} and U_{11}, U_{13} reside on grid-0; and L_{22}, L_{32} and U_{22}, U_{23} reside on grid-1. The factored block L_{33} and U_{33} reside only on grid-0. The right-hand side b_1 and b_2 reside on grid-0 and grid-1, respectively, whereas b_3 is replicated on both the process grids and initialized with zeros on grid-1. Figure 5 shows the timeline of SpTrS3D involving the L and uSolve substeps.

3.1.1 lSolve. In the lSolve, both grid-0 and grid-1 solves $L_{11}y_1 = b_1$ and $L_{22}y_2 = b_2$ in parallel, and update corresponding b_3 blocks as

$$b_3^0 = b_3^0 - L_{31}y_1$$

on grid-0, and

$$b_3^1 = -L_{32}y_2$$

on grid-1. After the update, grid-1 sends the b_3^1 to grid-0, which accumulates the updates on b_3 from both grids as follows:

$$b_3^0 = b_3^0 + b_3^1 = b_3^0 - L_{31}y_1 - L_{32}y_2.$$

Thus, the updated b_3^0 contains updates from both process grids, and then grid-0 solves $L_{33}y_3 = b_3$ for the final y_3.

3.1.2 uSolve. The uSolve can start after grid-0 has computed y_3. First, grid-0 solves $U_{33}x_3 = y_3$ for x_3 and sends x_3 to grid-1. Now, using x_3, both grid-0 and grid-1 can update the $y_1 = y_1 - U_{13}x_3$ and $y_2 = y_2 - U_{23}x_3$ respectively. Lastly, grid-0 and grid-1 solve $U_{11}x_1 = y_1$ and $U_{22}x_2 = y_2$ for x_1 and x_2 respectively. So, at the end of L and U solve, the final solution x_1 and x_2 reside in grid-0 and grid-1, and x_3 is replicated in both process grids. The communication pattern in uSolve is reverse of lSolve.

3.2 A more general case

The 3D sparse LU factorization algorithm uses $P_z = 2^l$ 2D grids [21]. The triangular solve can be extended for $P_z = 2^l$ in a similar fashion. In subsequent discussion, we focus on lSolve since, qualitatively, U- and lSolves have same structure, albeit in a reverse order.

In the lSolve, each two grid performs the lSolve for its leaf-subtree and accumulates update on b_k's, for each supernode k in its ancestor subtrees. Before performing lSolve for ancestor subtree,

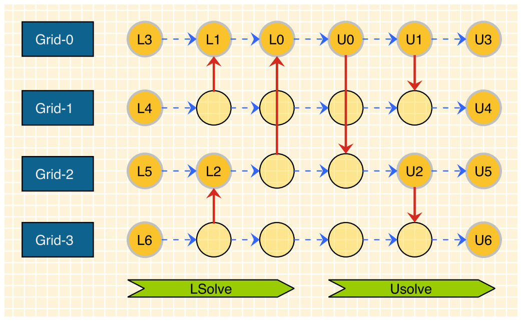


Figure 5: Timeline (from left to right) of SpTrS3D for $P_z = 2^l$, $l = 2$ two-dimensional process grids. Here each node with label Lk or Uk denotes a 2D triangular solve, $L_{kk}y_k = b_k$ or $U_{kk}x_k = y_k$. A red arrow denotes communication and direction between two process grids.

updates on b_k from different subtrees are reduced to a 2D grid, and the 2D performs the lSolve in the 2D fashion.

For instance, in Fig. 4, the etree is partitioned for $P_z = 2^2$ 2D grids numbered 0 to 3. In the first step, each of the 2D grids performs lSolve on the leaf subtrees (node-3 to 6) and performs the updates on respective ancestor subtrees. In the second step, grids 0 and 1 reduce the update on node-1 to grid-0, and grid-0 performs the lSolve for node-1; and grids 2 and 3 reduce the update on node-2 to grid-2, and grid-2 performs the lSolve for node-2. Both grids 0 and 2 perform the updates on node-0, the root of the tree. In the final step, updates on node-0 from all the grids are reduced to grid-0 and grid-0 performs the lSolve in 2D fashion.

The uSolve starts right after grid-0 has finished lSolve for node-0, and then grid-0 performs uSolve for node-0 and broadcasts it to all the grids so each process can perform the local-update. In the second step, grid-0 and grid-2 performs the uSolve for node-1 and 2, respectively, followed by broadcasting it to grid-1 and 3. Finally, each grid performs the uSolve for their respective leaf subtrees.

The 3D lSolve is shown in Algorithm 2. Figure 5 illustrates the timeline for SpTrS3D when there are $P_z = 4$ 2D grids.

4 COMMUNICATION ANALYSIS

We analyze the communication costs and volume of the SpTrS3D for triangular matrices that occur in solving PDEs with two- and three-dimensional geometries. We assess three communication metrics:

- *Communication cost*, W, which is the number of words sent along the critical path of the computation.
- *Average per-process communication volume*, V^{avg}, which is the average data sent among all the processes.
- *Maximum per-process communication volume*, V^{max}, which is the maximum number of data sent by any process.

The difference between communication cost and volume can be better understood with the following example. Consider a ring broadcast of data of length γ units between P processes, i.e., p_0 sends a message of length γ to p_1, which then relays it to p_2 and so on, until all the P processes have received the message. The time to finish the broadcast (T_{comm}) will be $(\alpha + \beta\gamma)(P - 1)$, where α is

Algorithm 2 3D Sparse Lower Triangular Solve Algorithm

```
Require: Factored $L$ and $U$ matrices, $b$: right hand side; Process coordinates $\{p_x, p_y, p_z\}$; $E_f$: grid-local etree; $p_z = 2^l$ for some integer $l$
    LSOLVE: $y \leftarrow L^{-1}b$
 1: for lvl in $l$ : 0 do:                                  ▷ Bottom-up traversal of $E_f$
 2:   if $p_z = k2^{l-\text{lvl}}$, $k \in \mathbb{Z}$ then:
 3:     $\sigma \leftarrow E_f[\text{lvl}]$                 ▷ $\sigma$ is the index of subtree
 4:     $y_\sigma \leftarrow$ LSOLVE2D($L_\sigma, b_\sigma$)
 5:     $b_i \leftarrow b_i - \sum_{i \in Anc(\sigma)} L_i y_\sigma$      ▷ Local-update
 6:     if lvl > 0 then:
 7:       if $k$ mod $2 \equiv 0$ then:                     ▷ Note $p_z = k2^{l-\text{lvl}}$
 8:         dest = $p_z$
 9:         src = $p_z + 2^{l-\text{lvl}}$
10:       else:
11:         src = $p_z$
12:         dest = $p_z - 2^{l-\text{lvl}}$
13:       for $l_a$ in lvl − 1 : 0 do:
14:         for $s \in E_f[l_a]$ do:
15:           if $p_z$ = src then:
16:             Send $b_s^{\text{src}}$ to dest
17:           else:
18:             Receive $b_s^{\text{src}}$ from src
19:             $b_s^{\text{dest}} = b_s^{\text{dest}} + b_s^{\text{src}}$
    return y
```

the cost of initiating a message transmission, and β is the cost of sending a unit data. The communication cost W is the coefficient of β in the expression for T_{comm}, i.e. $W = (P-1)\gamma$[1]. On the other hand, in this example V^{avg} will be $\gamma(P-1)/P$ and $V^{\text{max}} = \gamma$.

Informally, the communication cost W correlates to the time to completion when an application is communication-bound. The average per-process communication volume V^{avg} is a measure of energy spent in the communication and network load due to the computation; and V^{max} in an indicator of communication imbalance and possible network contention. In a dynamic asynchronous computation such as SpTrs, its difficult to precisely measure W, whereas V^{avg} and V^{max} can be measured readily, which is helpful in validating the analytical models that we develop in this section. Further, if a computation is entirely communication bound, then the following holds:

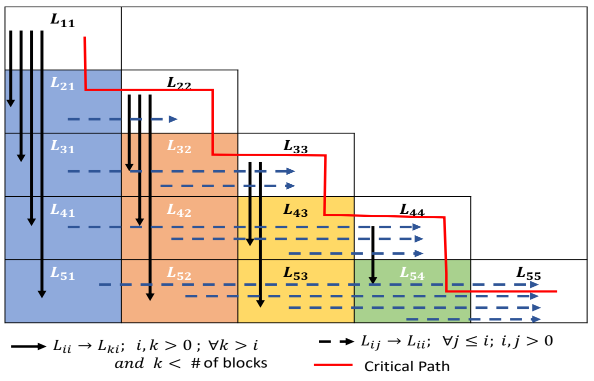
$$V^{\text{avg}} \leq V^{\text{max}} \leq W.$$

Thus, one can estimate a lower bound on W by using V^{max}. Hence V^{avg} and V^{max} provide an important insight into communication characteristics of any application.

4.1 Dense Triangular Solve on 2D Process Grid

Consider a dense lower triangular system $Ly = b$ distributed on a square 2D process grid of dimension $\sqrt{P} \times \sqrt{P}$ as shown in the Fig. 6. For sake of simplicity, we assume that blocking parameter for 2D block cyclic data distribution is one and number of right hand side is one i.e. $b \in \mathbb{R}^n$.

[1]We use #words as the unit for communication cost instead of time. This choice also facilitates direct comparison of communication cost and volume



L_{11}
L_{21}
L_{22}
L_{31}
L_{32}
L_{33}
L_{41}
L_{42}
L_{43}
L_{44}
L_{51}
L_{52}
L_{53}
L_{54}
L_{55}
$L_{ii} \rightarrow L_{ki}$; $i, k > 0$; $\forall k > i$ and $k <$ # of blocks
$L_{ij} \rightarrow L_{ii}$; $\forall j \leq i$; $i, j > 0$
Critical Path

Figure 6: Communication pattern in dense LSOLVE in 2D grid

4.1.1 *Communication Cost W.*

The critical path for the L solve is shown in Fig. 6. In the k-step of dense L solve, process P_{kk} computes the y_k and broadcast it across the process column $P_c(k)$. The process $P_{k+1,k}$ computes $b_{k+1}{-}{=}l_{k+1,k}y_k$ and sends it to the process $P_{k+1,k+1}$, which then computes y_{k+1}. Thus the total number of messages sent in the critical path of L-solve is $2(n-1)$, and each message has length γ. So in the case of dense L solve total communication cost in the critical path is given by:

$$W_{Dense}(n, P) = O(n)\,. \tag{2}$$

From Eq. (2), the communication cost in the dense L-solve in 2D process grid does not scale with the number of processors.

4.1.2 *Communication Volume V.*

In the dense L solve, each process only sends and receives $O\left(\frac{n}{\sqrt{P}}\right)$ words. So the per-process communication volume, in this case, is given by:

$$V_{Dense}(n, P) = V^{\text{avg}}_{Dense}(n, P) = V^{\text{max}}_{Dense}(n, P) = O\left(\frac{n}{\sqrt{P}}\right). \tag{3}$$

4.2 Planar Sparse Matrices

4.2.1 *2D Sparse Triangular Solve.*

In the case of planar sparse matrices, the top level separator is a dense matrix of dimensions $O(\sqrt{n})$. So the cost solving the top separator will be $W_{Dense}(\sqrt{n}) = O(\sqrt{n})$. In the first level, we have two separators of dimension $O\left(\sqrt{n/2}\right)$. Since solving the two separators in this level is independent, and is done in parallel, therefore communication costs will be $W_{Dense}(\sqrt{n/2}) = O\left(\sqrt{n/2}\right)$. The 2D triangular solve can exploit the parallelism of degree up to $\sqrt{P}$. So for the triangular solve of any level-i such that $2^i \leq \sqrt{P}$, the communication cost will be $W_{Dense}(\sqrt{n/2^i}) = O\left(\sqrt{n/2^i}\right)$. Let lvl_0 be the first level where $2^{lvl_0} > \sqrt{P}$, i.e.,

$$lvl_0 = \min\left\{i \mid 2^i > \sqrt{P},\ i \in \mathbb{Z}\right\} = \left\lceil \log_2 \sqrt{P} \right\rceil. \tag{4}$$

So $lvl_0 \approx 1/2 \log P$. We can write the total communication cost of triangular solve from level-0 to level-($lvl_0 - 1$) as:

$$W_{l<lvl_0}(n, P) = \sum_{i=0}^{lvl_0-1} \sqrt{\frac{n}{2^i}} = O(\sqrt{n}) \tag{5}$$

For levels $> lvl_0$, the 2D algorithm can exploit the $\sqrt{P}$ parallelism. The total number of variables in levels $> lvl_0$ is $n - \sqrt{n}P^{1/4} = O(n)$. Hence the total communication cost in solving levels$>lvl_0$ is

$$W_{l \geq lvl_0}(n, P) = \frac{n - \sqrt{n}P^{1/4}}{\sqrt{P}} = O\left(\frac{n}{\sqrt{P}}\right). \tag{6}$$

From Eqs. (5) and (6), the total communication cost for the 2D algorithm for the planar problems is given by:

$$W_{2D}(n, P) = O\left(\frac{n}{\sqrt{P}} + \sqrt{n}\right). \tag{7}$$

Communication Volume. To calculate the communication volume of the 2D algorithm, the sparse triangular system can be considered as a sequence of dense triangular systems of supernodes of dimension n_i so that $\sum_i n_i = n$. Since in the case of dense triangular solve in 2D process grid $V^{\text{avg}} = V^{\text{max}}$ (from Eq. (3)), it will be the same in this case as well. So the communication volume can be written as follows:

$$V_{2D}(n, P) = \sum_i V_{Dense}(n_i, P) = \frac{\sum_i n_i}{\sqrt{P}} = O\left(\frac{n}{\sqrt{P}}\right). \tag{8}$$

4.2.2 3D Sparse Triangular Solve. For the 3D algorithm, we have $P = P_z P_{xy}$, where P_z is the number of 2D grids each with P_{xy} processes. The 3D algorithm uses P_z is a power of two, $P_z = 2^{l_z}$. In our analysis, we assume that the 2D grid is a square grid of dimension $\sqrt{P_{xy}} \times \sqrt{P_{xy}}$.

We consider the communication costs of any process in grid-0, since it lies in the critical path of the triangular solve. The leaf subtree in grid-0 has dimension $\approx n/P_z$. The leaf-subtreee is solved by the 2D algorithm on a process grid of size P_{xy}. From Eq. (7), the communication costs of solving the leaf-subtree is:

$$\begin{aligned} W_{3D-leaf} = W_{2D}\left(\frac{n}{P_z}, P_{xy}\right) &= O\left(\frac{n}{P_z\sqrt{P_{xy}}} + \sqrt{\frac{n}{P_z}}\right) && (9) \\ &= O\left(\frac{n}{\sqrt{P_z P}} + \sqrt{\frac{n}{P_z}}\right). && (10) \end{aligned}$$

In each level-i, from 0 to $l_z - 1$, the grid-0 solves a dense triangular system of size $\sqrt{n/2^i}$, which has a communication cost of $W_{Dense}(\sqrt{n/2^i}, P_{xy}) = \sqrt{n/2^i}$. Thus the total communication cost in solving from level-0 to $l_z - 1$ is given by:

$$W_{3D-Anc}(n, P) = \sum_{i=0}^{l_z-1} \sqrt{\frac{n}{2^i}}. = O(\sqrt{n}) \tag{11}$$

Lastly, before solving any level-i from 0 to $l_z - 1$, grid-0 reduces the contribution from the other grids. In the i-th level, it receives vector of size $\sqrt{n/2^i}$. However, only the diagonal processes participate in this step. Hence the per-process communication cost for the reduction step in the i-th level is $\sqrt{\frac{n}{P_{xy}2^i}} = \sqrt{\frac{nP_z}{P2^i}}$. So the total communication cost in the reduction step from all the level is:

$$W^z(n, P, P_z) = \sum_{i=0}^{l_z-1} \sqrt{\frac{n}{P_{xy}2^i}} = O\left(\sqrt{\frac{nP_z}{P}}\right). \tag{12}$$

Combining Eqs. (10) to (12), we obtain the following expression for the communication cost of the 3D algorithm for planar matrices:

$$W_{3D}(n, P, P_z) = O\left(\frac{n}{\sqrt{P_z P}} + \sqrt{\frac{n}{P_z}} + \sqrt{n} + \sqrt{\frac{nP_z}{P}}\right). \tag{13}$$

Since $\sqrt{n} > \sqrt{\frac{n}{P_z}}$ and $\sqrt{n} > \sqrt{\frac{nP_z}{P}}$, hence we can simplify Eq. (13) to get the following expression:

$$W_{3D}(n, P, P_z) = O\left(\frac{n}{\sqrt{P_z P}} + \sqrt{n}\right) \tag{14}$$

Communication Volume. To get the average communication cost V^{avg}, it is sufficient to assume that each grid is solving a triangular solve of dimension n/P_z by using the 2D algorithm. Hence,

$$V^{\text{avg}}_{3D}(n, P) = V^{\text{avg}}_{2D}\left(\frac{n}{P_z}, P_{xy}\right) = O\left(\frac{n}{\sqrt{P_z P}}\right). \tag{15}$$

To calculate maximum per-process communication volume V^{max}, we consider the communication of any process in grid-0 since it participates in the all the level of triangular solve. The communication volume for any process in grid-0 has two components a) leaf-subtree solve which amount to $O\left(\frac{n}{\sqrt{P_z P}}\right)$; and b) ancestor-subtree solve, which has the same asymptotic complexity as solving top-level separator of dimension $\sqrt{n}$ in 2D grid of size P_{xy}, i.e. $V_{Dense}(\sqrt{n}, P_{xy}) = \frac{\sqrt{n}}{\sqrt{P_{xy}}} = O\left(\frac{\sqrt{nP_z}}{\sqrt{P}}\right)$. Thus, we can write the maximum per-process communication of the 3D algorithm as:

$$V^{\text{max}}_{3D}(n, P, P_z) = O\left(\frac{n}{\sqrt{P_z P}} + \frac{\sqrt{nP_z}}{\sqrt{P}}\right) \tag{16}$$

To minimize $V^{\text{max}}_{3D}(n, P)$, we should have $P_z = n^{1/2}$, in which case $V^{\text{max}}_{3D}(n, P) = O\left(\frac{n^{3/4}}{\sqrt{P}}\right)$. Hence optimal $V^{\text{max}}_{3D}(n, P)$ is smaller by a factor of $n^{1/4}$ to $V^{\text{max}}_{2D}(n, P)$.

4.3 Non-planar Sparse Matrices

In the case of non-planar sparse matrices, the top level separator has dimension $n^{2/3}$, and nodes in the i-th level have dimension $(n/2^i)^{2/3}$.

4.3.1 2D Sparse Triangular Solve. Similar to planar case, to calculate the communication costs of the 2D algorithm, we calculate $W_{l<lvl_0}(n, P)$ and $W_{l \geq lvl_0}(n, P)$, where lvl_0 is defined by Eq. (4). The corresponding equation to Eq. (5) for non-planar case can be written as:

$$W_{l<lvl_0}(n, P) = \sum_{i=0}^{lvl_0-1} \left(\frac{n}{2^i}\right)^{2/3} = O\left(n^{2/3}\right), \tag{17}$$

and equation corresponding to Eq. (6) is :

$$W_{l \geq lvl_0}(n, P) = \frac{n - n^{2/3}P^{1/4}}{\sqrt{P}} = O\left(\frac{n}{\sqrt{P}}\right). \tag{18}$$

So the total communication cost is given by:

$$W_{2D}(n, P) = O\left(\frac{n}{\sqrt{P}} + n^{2/3}\right) \tag{19}$$

Communication Volume. Eq. (8) also holds for non-planar input problems.

Table 2: Asymptotic communication cost and volume for SpTrS2D and SpTrS3D, on planar (2D PDE) and non-planar (3D-PDE) input problems

Problem type	Communication Param	SpTrS2D	SpTrS3D
Planar (2D PDE)	Cost (W)	$O\left(\frac{n}{\sqrt{P}}+\sqrt{n}\right)$	$O\left(\frac{n}{\sqrt{P_zP}}+\sqrt{n}\right)$
	Average Volume (V^{avg})	$O\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n}{\sqrt{P_zP}}\right)$
	Max Volume ($V^{\max}$)	$O\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n}{\sqrt{P_zP}}+\frac{\sqrt{nP_z}}{\sqrt{P}}\right)$
Non-Planar (3D PDE)	Cost (W)	$O\left(\frac{n}{\sqrt{P}}+n^{2/3}\right)$	$O\left(\frac{n}{\sqrt{P_zP}}+n^{2/3}\right)$
	Average Volume (V^{avg})	$O\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n}{\sqrt{P_zP}}\right)$
	Max Volume ($V^{\max}$)	$O\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n}{\sqrt{P_zP}}+n^{2/3}\frac{\sqrt{P_z}}{\sqrt{P}}\right)$

4.3.2 3D Sparse Triangular Solve. Similar to planar case, we calculate $W_{3D-leaf}$, W_{3D-Anc} and W^z for non-planar problems as follows:

$$W_{3D-leaf} = W_{2D}\left(\frac{n}{P_z}, P_{xy}\right) = O\left(\frac{n}{\sqrt{P_zP}} + \sqrt{\frac{n}{P_z}}\right) \tag{20}$$

$$W_{3D-Anc}(n,P) = \sum_{i=0}^{l_z-1}\left(\frac{n}{2^i}\right)^{2/3} = O\left(n^{2/3}\right) \tag{21}$$

$$W^z(n,P,P_z) = \sum_{i=0}^{l_z-1}\left(\frac{n}{P_{xy}2^i}\right)^{2/3} = O\left(\left(\frac{nP_z}{P}\right)^{2/3}\right) \tag{22}$$

$$\tag{23}$$

Combining Eqs. (20) to (22), we get the following expression for communication cost:

$$W_{3D}(n,P,P_z) = O\left(\frac{n}{\sqrt{P_zP}} + n^{2/3}\right) \tag{24}$$

Communication Volume. The expression for V_{3D}^{avg} for planar input problem Eq. (15) also hold for non planar problems. Using a similar argument as for the case of planar problems, we arrive at following expression for $V_{3D}^{\max}$ for non-planar problems

$$V_{3D}^{\max}(n,P,P_z) = O\left(\frac{n}{\sqrt{P_zP}} + n^{2/3}\frac{\sqrt{P_z}}{\sqrt{P}}\right) \tag{25}$$

To minimize communication volume, we should have $P_z = n^{1/3}$, in which case $V_{3D}^{\max}(n,P) = O\left(\frac{n^{5/6}}{\sqrt{P}}\right)$. Hence optimal $V_{3D}^{\max}(n,P)$ is smaller by a factor of $n^{1/6}$ to $V_{2D}^{\max}(n,P)$.

In Table 2, we summarize the asymptotic communication cost and volume for SpTrS2D and SpTrS3D on planar and non-planar input problems. In Section 5.4, we present some empirical result on average and maximum per-process communication volume.

5 RESULTS

In this section, we present results from a series of numerical experiment to understand the scalability of 3D sparse triangular solver algorithm.

Table 3: Test sparse matrices used in experiments

Name	Application	n	$\frac{nnz}{n}$
atmosmodd	CFD	1.3e6	6.9
boneS10	Model reduction	9.1e5	44.7
CurlCurl_4	Model Reduction	2.4e+6	10.9
dielFilterV3real	FEM/EM	1.1e+6	81.0
ldoor	Structural	9.5e+5	44.6
nlpkkt80	KKT matrices	1.1e+6	26.5
Ecology1	Ecology/Circuit	1.0e+6	5.0
S2D9pt3072	PDE	9.4e+6	9.0
Serena	Structural	1.4e+6	46.1
torso3	PDE	2.6e5	17.1

5.1 Experimental Set-up

5.1.1 Test Bed. We ran our experiments on a Cray XC30 machine "Edison" cluster at NERSC.[2] Each node of Edison contains dual-socket 12-core Intel Ivy Bridge processors. We chose the SuperLU_Dist's default parameters for running experiments, which is tuned for factorization phase. We used 4 OpenMP threads per MPI process with hyperthreading disabled. We compiled our code with Intel C compiler version 18.0.0 and linked with Intel MKL version 2017.2.174 for BLAS operations.

5.1.2 Test Matrices. We used a mix of planar and non-planar test matrices coming from different real world applications to evaluate the performance of 3D sparse triangular solver. The test matrices are listed in Table 3. The planar matrices come from the discretization of two-dimensional PDE s2D9pt2048) and circuit analysis (Ecology1). Five of the six non-planar matrices are from the discretization of 3D PDEs and one, matrix nlpkkt80, comes from non-linear optimization. The solve time for 16 right hand sides ranges from .5-10 seconds on 16 nodes when using the baseline 2D SuperLU_Dist.

5.2 Results on 16 nodes

On 16 nodes of the Edison cluster, the 3D sparse triangular solve configurations achieve 1.3-4.3× and 0.9-2.9× speedup with respect to 2D configuration for planar and non-planar matrices, respectively. The results appear in Fig. 7, which shows the factorization time normalized by the baseline 2D SuperLU_Dist for each matrix and process configuration. Columns correspond to different 3D process configurations. The leftmost column, $P_z = 1$, is the 2D algorithm; subsequent columns correspond to P_z values of 2, 4, 8, and 16. The factorization time is divided into two components, T_{comp} and T_{comm}. The T_{comp} is the time spent in local computation on the critical path of the combined L and U solve, and T_{comm} is the non-overlapped communication and synchronization time.

5.3 Strong Scaling

We now analyze the performance of 3D sparse triangular solver for different $P_{xy} \times P_z$ combinations for different number of right hand sides. For this experiment, we choose one planar matrix s2D9pt2048

[2]http://www.nersc.gov/users/computational-systems/edison

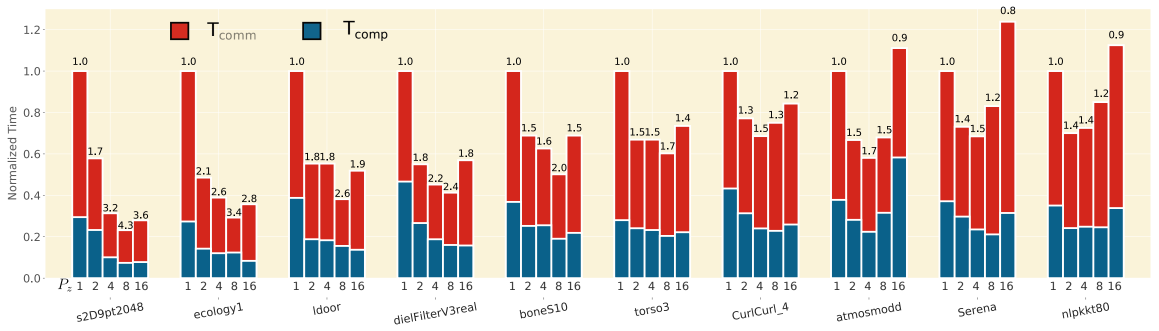


Figure 7: **The Triangular Solve performance for 16 right hand sides for various $P_x \times P_y \times P_z$ grids on 16 nodes (384 CPU cores) of the Edison system at NERSC. For each matrix, each column represents a different value of $P_z = \{1,\ 2,\ 4,\ 8,\ 16\}$ from left to right. Thus, the leftmost column is the $2D$ algorithm, and when moving right, the $2D$ grids become smaller as P_z increases. For each data set, the time shown is normalized with respect to $2D$ SuperLU_Dist on 16 nodes. T_{comp} represents the time spent in the local computation on the critical path, whereas T_{comm} is the non-overlapped time spent in communication and synchronization.**

and a non-planar matrix nlpkkt80. Let γ denotes the number of right hand sides.

$$\gamma = \#\text{ Right hand sides.}$$

We choose three different number of right hand sides $\gamma \in \{1, 16, 64\}$ for this experiment.

Strong scaling for s2D9pt2048. We show the results for s2D9pt2048 on Fig. 8. When $\gamma = 1$, the best case 3D configuration is 4.7× faster than best case 2D process configuration. When $\gamma = 1$, each message sent is short, thus the performance of across different configuration is limited by the latency costs than the bandwidth cost. For the 2D process configurations, the performance does not scale well with increasing grid size. This reflects that despite enough parallelism post-ordering, block-cyclic data distribution on non-square grids may not distribute the load evenly. Therefore, the solve-phase remains predominantly sequential. Since 3D configurations do not suffer from these limitations, so solve-phase shows some scalability with increasing P_z.

For $\gamma = 16$, the best case 3D configuration is 7× faster than best case 2D process configuration. In this case, 2D process configurations, the performance is limited by data transfer costs and scales as $O\left(1/\sqrt{P}\right)$. Again in this case, for a small value of P_z performance scales linearly and after certain P_z for a given 2D grid size, adding more 2D grids do not result in any further performance gains.

The case $\gamma = 64$ is similar to the case $\gamma = 16$. In this case, the 3D configuration is again approximately 7× than the best case 2D configuration. In this case, we can exploit efficient BLAS-3 calls effectively for local computation. Moreover, in this case, the fraction of computation is significantly more than either data transfer or latency cost. Hence, we achieve higher performance in this case for any process configuration.

Strong scaling for nlpkkt80. We show the strong scaling results for nlpkkt80 on Fig. 9 for $\gamma = 1,\ 16\&\ 64$.

When $\gamma = 1$, the 3D configuration achieves a best case speed up of 1.89× over 2D configurations. Similar to the case of s2D9pt2048 when $\gamma = 1$ performance of nlpkkt80 is limited by latency costs. However, since nlpkkt80 is a non-planar matrix, the latency costs increase more quickly compared to the planar case. For $\gamma = 16$ and $\gamma = 64$, the best case 3D configuration achieves a best case speed-up of 2.3× and 2.6× respectively.

In both cases, $\gamma = 16$ and $\gamma = 64$, we were able to scale to 24K cores of Edison, with continued improvement in performance.

5.4 Communication Volume

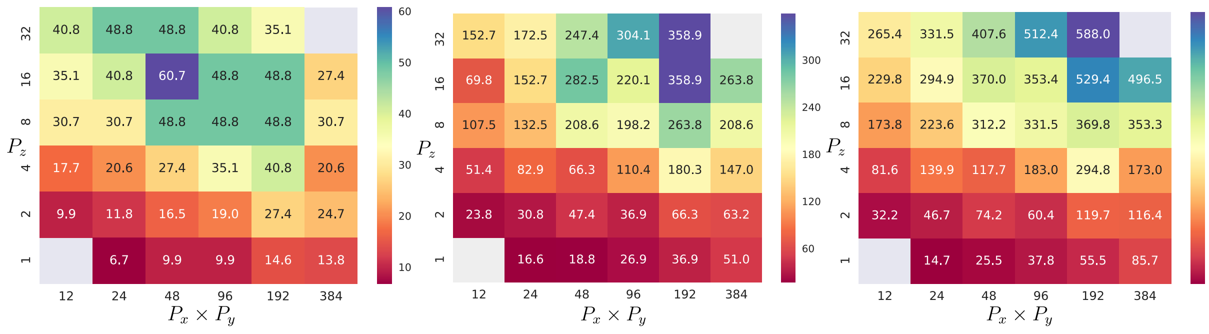
In Figs. 10 and 11, we show average and maximum per-process communication volume for s2d9pt2048 and nlpkkt80 on 96 and 384 MPI processes for $P_z \in \{1,\ 2,\ 4,\ 8,\ 16\}$ and $\gamma = 16$. The communication is divided into communication along xy-plane (shown in blue) and communication along z dimension (shown in red).

For both the matrices, the average per-process communication volume V (Figs. 10a and 11a) reduces as $\frac{1}{\sqrt{P_z}}$ for different P_z and constant total number of processes P. Similarly, V decreases as $\frac{1}{\sqrt{P}}$ with increasing P and constant P_z. Thus, we see a reduction of roughly 2× in average per-process communication volume when we go from $P = 96$ to $P = 384$. This agrees with our models for communication volume described in Eqs. (16) and (25). In all the cases, communication volume along z-dimension is a tiny fraction of total communication.

The maximum per-process communication volume for the 2D algorithm (Figs. 10b and 11b) is 2.3× the average communication volume, indicating some communication imbalance. The 3D configurations, besides reducing average per-process communication, also attenuate the communication imbalance, e.g. at 96 processors $P_z = 2$ maximum per-process communication is 1.4 and 1.42× the average per-process communication for s2d9pt2048, and nlpkkt80; whereas for the 2D algorithm ($P_z = 1$), the ratio of maximum versus average per-process communication is 2.2 and 2.3× for s2d9pt2048, and nlpkkt80, respectively.

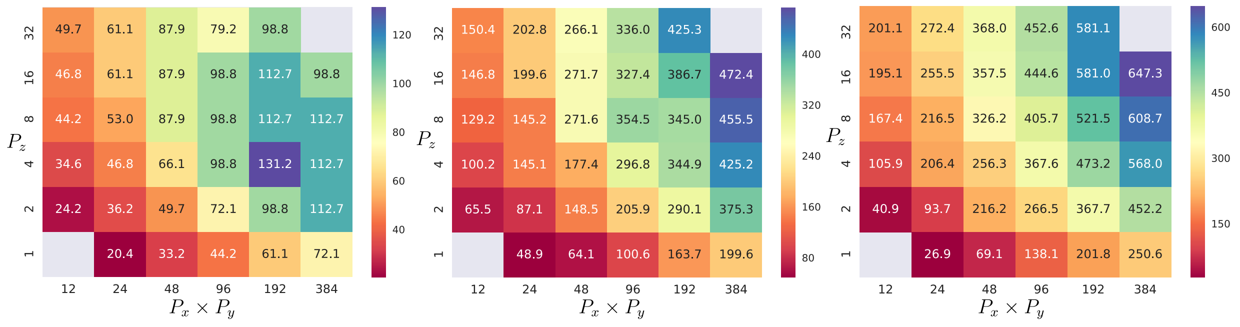
6 RELATED WORK

Complementary to our approach of reducing communication by employing a 3D process grid, researchers have looked into selective inversion [11, 19, 22], re-ordering to adapt to structure [23], and improving performance of collective operations [17]. Multifrontal methods with the so-called subtree-to-subcube mapping [7] also



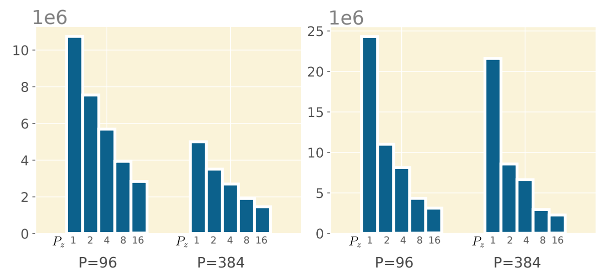
(a) $\gamma = 1$ **(b)** $\gamma = 16$ **(c)** $\gamma = 64$

Figure 8: The triangular solve performance (in Gigaflop/s) for different number of *right hand sides* (γ) for different $P_{xy} \times P_z$ for planar matrix s2D9pt2048.



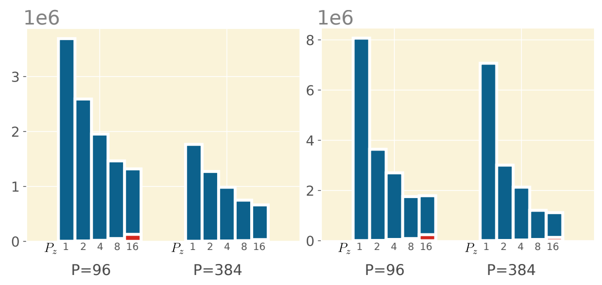
(a) $\gamma = 1$ **(b)** $\gamma = 16$ **(c)** $\gamma = 64$

Figure 9: The triangular solve performance (in Gigaflop/s) for different number of *right hand sides* (γ) for different $P_{xy} \times P_z$ for non-planar matrix nlpkkt80.



(a) Average per-process communication **(b) Maximum per-process communication**

Figure 10: Per-processs Communication Volume for s2d9pt2048: Fig. 10a shows the average per-process communication volume for 96 (left) and 384 (right) MPI processes for different P_z; Fig. 10b shows the maximum per-process communication volume for 96 (left) and 384 (right) MPI processes for different P_z.



(a) Average per-process communication **(b) Maximum per-process communication**

Figure 11: Per-processs Communication Volume for nlpkkt80: Fig. 11a shows the average per-process communication volume for 96 (left) and 384 (right) MPI processes for different P_z; Fig. 11b shows the maximum per-process communication volume for 96 (left) and 384 (right) MPI processes for different P_z.

use elimination tree parallelism to improve locality and reduce communication. One notable example is the method for Cholesky factorization by Gupta et al. [9], which describes an efficient triangular solver for such a mapping [14]. More comprehensive discussions on differences between right-looking and multifrontal methods appears elsewhere [10, 20].

For dense triangular solve, there also exist communication-avoiding algorithms that use 3D process grids [13, 24].

Communication-avoiding methods have been proposed for constructing Krylov Subspace for iterative solver [4, 12]. In theory, such techniques can be also applied for iterative solvers that use triangular preconditioners. For stationary iterations, researchers have explored asynchronous iterations to reduce synchronization costs [2, 3, 5].

Beyond the case of sparse linear solvers, machine learning algorithms on large and sparse data have renewed interest in communication efficient algorithms for other sparse matrix operations, leading to methods for sparse-times-dense matrix multiplication [16] and sparse-sparse matrix multiplication [1, 8, 25], to name a few.

7 CONCLUSION

This paper extends our 3D data structure for sparse LU factorization [21] to sparse triangular solve. Our analysis shows that the resulting SpTrs also becomes communication-avoiding.

A better SpTrs like ours can lead to a better overall direct solver. At present, SuperLU_Dist3D factors the matrix using a 3D process grid of size $P_x \times P_y \times P_z$ and then gathers the LU factors into a 2D of dimension $P_x \times P_y$ to perform its SpTrs. By contrast, our new 3D triangular solve eliminates the need for gathering the L and U factors, enabling the use of all $P_x \times P_y \times P_z$ processors. Besides mitigating such an inefficiency, the 3D SpTrs improves on the asymptotic communication costs of the 2D algorithm. Thus, while this paper focuses on SpTrs, complete integration into the full direct solver is an important next step.

Despite these improvements, the dense triangular solve that occurs in the ancestor subtrees is not fully parallel, leading to $O\left(\sqrt{n}\right)$ and $O\left(n^{2/3}\right)$ terms in the communication costs for SpTrS3D on 2D and 3D problems. That does not scale with the number of processors. Since the dimension of the ancestor subtrees is smaller than the dimension of the problem by an order of magnitude, a different strategy is needed. In particular, one ought to consider computing the inverses of dense L and U factors of the ancestor-subtrees and perform matrix-vector multiplication with L^{-1} and U^{-1} instead of performing a triangular solve. These inverses can be computed during the process of factorization without any additional communication-overhead, and will increase computation and memory at most by a factor of two. We plan to investigate the feasibility of this approach in the future.

Prior to this work, much of the work in communication-avoiding sparse and dense linear algebra was limited to BLAS Level-3 style matrix-matrix type operations. This work presents what might be one of the first cases of using communication-avoiding techniques and 3D process grids for sparse *matrix-vector* operations, or BLAS Level-2. However, sparse triangular matrices in the direct solver have significantly more non-zeros per-row, e.g. $O(\log n)$, $O\left(n^{1/3}\right)$, for 2D and 3D problems respectively, than general sparse matrices, which typically have $O(1)$ non-zeros per row. Nevertheless, the idea of using nested-dissection-type 3D data distributions can in principle be extended to other sparse BLAS Level-2 and Level-3 operations, such as distributed sparse matrix time dense vector/matrix multiplication, sparse-sparse matrix multiplication, sparse QR factorization, and graph algorithms such as breadth-first search and all pair shortest path. Determining the efficacy of combining nested-dissection and 3D data distribution for other sparse problems is another avenue for future investigation.

REFERENCES

[1] Ariful Azad, Grey Ballard, Aydin Buluc, James Demmel, Laura Grigori, Oded Schwartz, Sivan Toledo, and Samuel Williams. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. *SIAM Journal on Scientific Computing*, 38(6):C624–C651, 2016.

[2] Gérard M Baudet. Asynchronous iterative methods for multiprocessors. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1976.

[3] Edmond Chow. Convergence models and surprising results for the asynchronous Jacobi method. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 940–949. IEEE, 2018.

[4] James Demmel, Mark Hoemmen, Marghoob Mohiyuddin, and Katherine Yelick. Avoiding communication in sparse matrix computations. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12. IEEE, 2008.

[5] Andreas Frommer and Daniel B Szyld. On asynchronous iterations. *Journal of computational and applied mathematics*, 123(1-2):201–216, 2000.

[6] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[7] Alan George, Joseph WH Liu, and Esmond Ng. Communication results for parallel sparse cholesky factorization on a hypercube. *Parallel Computing*, 10(3):287–298, 1989.

[8] Amir Gholami, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. Integrated model, batch and domain parallelism in training neural networks. *arXiv preprint arXiv:1712.04432*, 2017.

[9] Anshul Gupta, George Karypis, and Vipin Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):502–520, 1997.

[10] Michael T Heath, Esmond Ng, and Barry W Peyton. Parallel algorithms for sparse linear systems. *SIAM review*, 33(3):420–460, 1991.

[11] Michael T Heath and Padma Raghavan. Performance of parallel sparse triangular solution. In *Algorithms for Parallel Processing*, pages 289–305. Springer, 1999.

[12] Mark Hoemmen. *Communication-avoiding Krylov subspace methods*. University of California, Berkeley, 2010.

[13] Dror Irony and Sivan Toledo. Trading replication for communication in parallel distributed-memory dense solvers. *Parallel Processing Letters*, 12(01):79–94, 2002.

[14] Mahesh V Joshi, Anshul Gupta, George Karypis, and Vipin Kumar. A high performance two dimensional scalable parallel algorithm for solving sparse triangular systems. In *High-Performance Computing, 1997. Proceedings. Fourth International Conference on*, pages 137–143. IEEE, 1997.

[15] George Karypis and Vipin Kumar. Family of graph and hypergraph partitioning software. http://glaros.dtc.umn.edu/gkhome/views/metis. Accessed: 2014-01-26.

[16] Penporn Koanantakool, Ariful Azad, Aydin Buluç, Dmitriy Morozov, Sang-Yun Oh, Leonid Oliker, and Katherine Yelick. Communication-avoiding parallel sparse-dense matrix-matrix multiplication. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 842–853. IEEE, 2016.

[17] Yang Liu, Mathias Jacquelin, Pieter Ghysels, and Xiaoye S Li. Highly scalable distributed-memory sparse triangular solution algorithms. In *2018 Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing*, pages 87–96. SIAM, 2018.

[18] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[19] Padma Raghavan. Efficient parallel sparse triangular solution using selective inversion. *Parallel Processing Letters*, 8(01):29–40, 1998.

[20] Edward Rothberg. Exploiting the memory hierarchy in sequential and parallel sparse Cholesky factorization. Technical report, Stanford University, Department of Computer Science, 1992.

[21] Piyush Sao, Xiaoye S. Li, and Richard Vuduc. A communication-avoiding 3D LU factorization algorithm for sparse matrices. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 908–919, Vancouver, BC, Canada, May 2018.

[22] Keita Teranishi, Padma Raghavan, and Esmond Ng. A new data-mapping scheme for latency-tolerant distributed sparse triangular solution. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 27–27. IEEE, 2002.

[23] Ehsan Totoni, Michael T Heath, and Laxmikant V Kale. Structure-adaptive parallel solution of sparse triangular linear systems. *Parallel Computing*, 40(9):454–470, 2014.

[24] Tobias Wicky, Edgar Solomonik, and Torsten Hoefler. Communication-avoiding parallel algorithms for solving triangular systems of linear equations. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, pages 678–687. IEEE, 2017.

[25] Carl Yang, Aydın Buluç, and John D Owens. Design principles for sparse matrix multiplication on the gpu. In *European Conference on Parallel Processing*, pages 672–687. Springer, 2018.