Communication-Avoiding Parallel Algorithms for Solving Triangular Systems of Linear Equations

Tobias Wicky
Department of Computer Science
ETH Zurich
Zurich, Switzerland
twicki@ethz.ch

Edgar Solomonik
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
solomon2@illinois.edu

Torsten Hoefler
Department of Computer Science
ETH Zurich
Zurich, Switzerland
htor@inf.ethz.ch

***Abstract*—We present a new parallel algorithm for solving triangular systems with multiple right hand sides (TRSM). TRSM is used extensively in numerical linear algebra computations, both to solve triangular linear systems of equations as well as to compute factorizations with triangular matrices, such as Cholesky, LU, and QR. Our algorithm achieves better theoretical scalability than known alternatives, while maintaining numerical stability, via selective use of triangular matrix inversion. We leverage the fact that triangular inversion and matrix multiplication are more parallelizable than the standard TRSM algorithm. By only inverting triangular blocks along the diagonal of the initial matrix, we generalize the usual way of TRSM computation and the full matrix inversion approach. This flexibility leads to an efficient algorithm for any ratio of the number of right hand sides to the triangular matrix dimension. We provide a detailed communication cost analysis for our algorithm as well as for the recursive triangular matrix inversion. This cost analysis makes it possible to determine optimal block sizes and processor grids a priori. Relative to the best known algorithms for TRSM, our approach can require asymptotically fewer messages, while performing optimal amounts of computation and communication in terms of words sent.**

***Keywords*-TRSM, communication cost, 3D algorithms**

I. Introduction

Triangular solve for multiple right hand sides (TRSM) is a crucial subroutine in many numerical linear algebra algorithms, such as LU and Cholesky factorizations [1], [2]. Moreover, it is used to solve linear systems of equations once the equation matrix is decomposed using any factorization involving a triangular matrix factor. We consider TRSM for dense linear equations in the matrix form,

$$L \cdot X = B,$$

where $L \in \mathbb{R}^{n \times n}$ is a lower-triangular matrix while $B \in \mathbb{R}^{n \times k}$, $X \in \mathbb{R}^{n \times k}$ are dense matrices. We study the communication cost complexity of two variants of the TRSM algorithm for parallel execution on p processors with unbounded memory. First, we present an adaptation of a known recursive scheme for TRSM [3] along with a complete communication cost analysis. Then, we demonstrate a new algorithm that uses selective triangular inversion to reduce the synchronization cost over known schemes, while preserving optimal communication and computation costs. Careful choice of algorithmic parameters, allows us to achieve better asymptotic complexity for a large (and most important) range of input configurations.

Our TRSM algorithm leverages matrix multiplication and triangular matrix inversion as primitives. We provide a communication-efficient parallel matrix multiplication algorithm that starts from a 2D cyclic distribution, a modest enhancement to existing approaches. Triangular matrix inversion provides the key ingredient to the lower synchronization cost in our TRSM algorithm. Unlike general matrix inversion, triangular inversion is numerically stable [4] and can be done with relatively few synchronizations. We present a known parallel approach for triangular inversion and provide the first communication cost analysis thereof.

We invert triangular diagonal blocks of the L matrix at the start of our TRSM algorithm, increasing the computational granularity of the main part of the solver. Inverting blocks, rather than the whole matrix, also allows our algorithm to be work-efficient in cases when the number of right-hand sides is smaller than the matrix dimension. We formulate the algorithm in an iterative, rather than a recursive manner, avoiding overheads incurred by the known parallel recursive TRSM approach. This innovation reduces communication cost by a factor of $\Theta(\log(p))$ relative to the recursive algorithm when the number of right-hand sides is relatively small. At the same time, across a large range of input parameters, we achieve a synchronization cost improvement over the recursive approach by a factor of $\Theta\left(\left(\frac{n}{k}\right)^{1/6} p^{2/3}\right)$.

II. Preliminaries

A. Execution Time Model

The model we use to calculate the parallel execution time of an algorithm along its critical path is the $\alpha - \beta - \gamma$ model. It describes the total execution time of the algorithm T in terms of the floating point operations (flop) count F, the bandwidth W (number of words of data sent and received)

and the latency S (number of messages sent and received) along the critical path [5] in the following fashion:

$$T = \alpha \cdot S + \beta \cdot W + \gamma \cdot F.$$

For all the terms we only show the leading order cost in terms of n, k and p. We assume that every processor can send and receive one message at a time in point to point communication. We do not place constraints on the local memory size.

B. Notation

For brevity, we will sometimes omit specification of the logarithm base, using $\log$ to denote $\log_2$. We will make frequent use of the unit step function

$$\mathbb{1}_x = \begin{cases} 1 : x > 1 \\ 0 : x \leq 1. \end{cases}$$

We use $\Pi(x_1, \ldots, x_n)$to denote single processors in an n-dimensional processor grids.

To refer to successive elements or blocks, we will use the colon notation where $i : j = [i, i+1, \ldots, j-1] \quad i < j$. To refer to strided sets of elements or blocks, we will write

$$i : k : j = [i, i+k, i+2k, \ldots, i+\alpha k] \quad \max \alpha \text{ s.t. } \alpha k < j.$$

The colon notation can be applied to a list and should there be considered element-wise. To use subsets of processors, we use the $\circ$ notation in the way that $\Pi(x, \circ, z) = \Pi(x, 1 : p_{\mathrm{y}}, z)$ denotes a (1-dimensional) grid of processors in the y-dimension.

Global matrices are denoted by capital letters whereas locally owned parts have square brackets: If L is distributed on $\Pi(\circ, \circ, 1)$, every processor $\Pi(x, y, 1)$ owns $L[x, y]$. Matrix elements are accessed with brackets.

C. Previous Work

We now cover necessary existing building blocks (e.g. collective communication routines) and previous work. In particular, we overview related results on communication cost of matrix multiplication and triangular solves.

1) Collective Communication: In [6], Chan et al. present a way to perform reduction, allreduction and broadcast via allgather, scatter, gather and reduce-scatter. The latter set of collectives can be done using recursive doubling (butterfly algorithms) [6], [7] for a power of two number of processors. If we have a non-power of two number of processors, the algorithm described in [8] can be used. For simplicity, we do not consider reduction and broadcast algorithms that can achieve a factor of two less in cost in specific regimes of α and β [9]. If we use butterfly methods, the cost of an allgather of n words among p processors is

$$T_{\mathrm{allgather}}(n, p) = \alpha \cdot \log p + \beta \cdot n\mathbb{1}_p.$$

Scatter and gather also have the same cost [7],

$$T_{\mathrm{scatter}}(n, p) = \alpha \cdot \log p + \beta \cdot n\mathbb{1}_p,$$
$$T_{\mathrm{gather}}(n, p) = \alpha \cdot \log p + \beta \cdot n\mathbb{1}_p.$$

Reduce-scatter uses the same communication-path and has same communication cost but we have to add the additional overhead of local computation,

$$T_{\mathrm{reduce-scatter}}(n, p) = \alpha \cdot \log p + \beta \cdot n\mathbb{1}_p + \gamma \cdot n\mathbb{1}_p.$$

The cost of an all-to-all among p processors is

$$T_{\mathrm{alltoall}}(n, p) = \alpha \cdot \log(p) + \beta \cdot \frac{n \log p}{2}.$$

The combination of the algorithms leads to the following costs for reduction, allreduction, and broadcast:

$$T_{\mathrm{reduction}}(n, p) = \alpha \cdot 2 \log p + \beta \cdot 2n\mathbb{1}_p + \gamma \cdot n\mathbb{1}_p,$$
$$T_{\mathrm{allreduction}}(n, p) = \alpha \cdot 2 \log p + \beta \cdot 2n\mathbb{1}_p + \gamma \cdot n\mathbb{1}_p,$$
$$T_{\mathrm{bcast}}(n, p) = \alpha \cdot 2 \log p + \beta \cdot 2n\mathbb{1}_p.$$

2) Matrix Multiplication: Communication-efficient parallel algorithms for matrix multiplication have been analyzed extensively [10]–[15]. In [16], Demmel et al. present algorithms that are asymptotically optimal for matrix multiplication of arbitrary (potentially non square) matrices. If we neglect the memory terms, their work shows that matrix multiplication can be done with the following costs.

Bandwidth: When multiplying a matrix A that is of dimension $n \times n$ with a matrix B of dimensions $n \times k$ with p processors, we obtain an asymptotic bandwidth cost of

$$W_{\mathrm{MM}}(n, k, p) = \begin{cases} \mathcal{O}\left(\frac{nk}{\sqrt{p}}\right) & n > k \cdot \sqrt{p} \\ \mathcal{O}\left(\left(\frac{n^2 k}{p}\right)^{2/3}\right) & k/p \leq n \leq k \cdot \sqrt{p} \\ \mathcal{O}\left(n^2\right) & n < k/p. \end{cases}$$

We refer to the first of the cases of W_{MM}, as the case of two large dimensions, here the matrix A is much larger than the right hand side B, when the best way of performing a matrix multiplication is to use a two dimensional layout for the processor grid. The second case, three large dimensions, has matrices A and B of approximately the same size. A three dimensional grid layout is optimal here. And the third case, one large dimension, is the case where the right hand side B is larger than the triangular matrix A, the best way to do a matrix multiplication is to use a one dimensional layout for the processor grid.

Latency: Assuming unlimited memory, matrix multiplication as presented in [16] can be done with a latency of

$$S_{\mathrm{MM}}(p) = \mathcal{O}\left(\log(p)\right).$$

Flop Cost: Matrix multiplication takes $\mathcal{O}\left(n^2 k\right)$ flops, which can be divided on p processors and therefore we have

$$F_{\mathrm{MM}}(n, k, p) = \mathcal{O}\left(\frac{n^2 k}{p}\right).$$

Previous Analysis: For the case where $k = n$ the bandwidth analysis of a general matrix multiplication goes back to what is presented in [12]. Aggarwal et al. present a cost analysis in the LPRAM model. In that work, the authors show that the same cost can also be achieved for the transitive closure problem that can be extended to the problem of doing an LU decomposition. The fact that these bandwidth costs can be obtained for the LU decomposition was later demonstrated by Tiskin [17]. He used the bulk synchronous parallel (BSP) execution time model. Since the dependencies in LU are more complicated than they are for TRSM, we also expect TRSM to be able to have the same asymptotic bandwidth and flop costs as a general matrix multiplication.

3) Triangular Matrix Solve for Single Right Hand Sides: Algorithms for the problem of triangular solve for a single right hand side (when X and B are vectors) have been well-studied. A communication-efficient parallel algorithm was given by Heath and Romine [18]. This parallel algorithm was later shown to be an optimal schedule in latency and bandwidth costs via lower bounds [5]. However, when X and B are matrices ($k > 1$), it is possible to achieve significantly lower communication costs relative to the amount of computation required. The application of selective inversion has been used to accelerate repeated triangular solves that arise in preconditioned sparse iterative methods [19].

4) Recursive Triangular Matrix Solve for Multiple Right Hand Sides: A recursive approach of solving the TRSM-problem was presented in the work of Elmroth et al. [3]. The initial problem, $L \cdot X = B$ can be split into $L \cdot \begin{bmatrix} X_1 & X_2 \end{bmatrix} = \begin{bmatrix} B_1 & B_2 \end{bmatrix}$, which yields two independent subproblems:

$$L \cdot X_1 = B_1, \qquad L \cdot X_2 = B_2.$$

hence the subproblems are independent and can be solved in parallel.

The other, dependent splitting proposed divides the triangular matrix, yielding two subtasks that have to be solved one at a time

$$\begin{bmatrix} L_{11} & \\ L_{12} & L_{22} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix},$$

where we obtain the dependent subproblems:

$$L_{11} \cdot X_1 = B_1 \quad \text{and} \quad L_{22} \cdot X_2 = B_2 - L_{12} \cdot X_1.$$

These problems are dependent as we need the solution X_1 to solve the second problem.

Parallel TRSM algorithms with 3D processor grids can reduce the communication cost in an analogous fashion to matrix multiplication. Irony and Toledo [20] presented the first parallelization of the recursive TRSM algorithm with a 3D processor grid. They demonstrated that the communication volume of their parallelization is $\mathcal{O}(nkp^{1/3} + n^2p^{1/3})$. Thus each processor communicates $\mathcal{O}((nk + n^2)/p^{2/3})$ elements, which is asymptotically equal to $W_{\mathrm{MM}}(n, k, p)$ when $k = \Theta(n)$. However, they did not provide a bound on the latency cost nor on the communication bandwidth cost along the critical path, so it is unclear to what extent the communication volume is load-balanced. Lipshitz [21] provides an analysis of the recursive TRSM algorithm in the same communication cost model as used in this paper. For the case of $k = n$, his analysis demonstrates

$$T_{\mathrm{TRSM-L}}(n, n, p) = \mathcal{O}(p^{2/3} \cdot \alpha + n^2/p^{2/3} \cdot \beta + n^3/p \cdot \gamma).$$

For some choices of algorithmic parameters, the analysis in [21] should lead to the bandwidth cost,

$$W_{\mathrm{TRSM-L}}(n, k, p) = \mathcal{O}\left(\left(\frac{n^2k}{p}\right)^{2/3}\right),$$

which is as good as matrix multiplication, $W_{\mathrm{MM}}(n, k, p)$. However, it is unclear how to choose the parameters of the formulation in [21] to minimize latency cost for general n, k. Prior to presenting our main contribution (an inversion-based TRSM algorithm with a lower latency cost), we provide a simpler form of the recursive TRSM algorithm and its asymptotic cost complexity. Inversion has previously been used in TRSM implementations [22], but our study is the first to consider communication-optimality. We start by presenting a subroutine for 3D matrix multiplication which operates from a starting 2D distribution, simplifying the subsequent presentation of TRSM algorithms.

III. Matrix Multiplication

$B = \mathbf{MM}(L, X, \Pi_{2D}, n, k, p, p_1, p_2)$

Require:

The processor grid Π_{2D} has dimensions $\sqrt{p} \times \sqrt{p}$
L is an $n \times n$ matrix, distributed on Π_{2D} in a cyclic layout, so $\Pi_{2D}(x, y)$ owns $L[x, y]$ of size $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ such that $L[x, y](i, j) = L(i\sqrt{p} + x, j\sqrt{p} + y)$.
X is a dense $n \times k$ matrix is distributed cyclically so that $\Pi_{2D}(x, y)$ owns $X[x, y]$ of size $\frac{n}{\sqrt{p}} \times \frac{k}{\sqrt{p}}$

1: Define a $p_1 \times \sqrt{p_2} \times p_1 \times \sqrt{p_2}$ processor grid Π_{4D}, such that $\Pi_{4D}(x_1, x_2, y_1, y_2) = \Pi_{2D}(x_1 + p_1x_2, y_1 + p_2y_2)$ owns blocks $L[x_1, x_2, y_1, y_2]$ and $X[x_1, x_2, y_1, y_2]$
2: $L'[x_1, y_1] = \mathbf{Allgather}\,(L[x_1, \circ, y_1, \circ], \Pi_{4D}(x_1, \circ, y_1, \circ))$
3: $X'[x_1, y_1, x_2, y_2]$
$= \mathbf{Transpose}(X[x_1, x_2, y_1, y_2], \Pi_{4D}(x_1, x_2, y_1, y_2), x_2, y_1)$
4: $X''[y_1, x_1, x_2, y_2]$
$= \mathbf{Transpose}(X'[x_1, y_1, x_2, y_2], \Pi_{4D}(x_1, x_2, y_1, y_2), x_1, y_1)$
5: $X'''[y_1, x_2, y_2] =$
$\mathbf{Allgather}(X''[y_1, \circ, x_2, y_2], \Pi_{4D}(\circ, x_2, y_1, y_2))$
6: $\Pi_{4D}(x_1, x_2, y_1, y_2)$:
$B''[x_1, y_1, x_2, y_2] = L'[x_1, y_1] \cdot X'''[y_1, x_2, y_2]$
7: $B'[x_1, y_1, x_2, y_2]$
$= \mathbf{Scatter\text{-}reduce}(B''[x_1, \circ, x_2, y_2], \Pi_{4D}(x_1, x_2, \circ, y_2))$
8: $B[x_1, x_2, y_1, y_2]$
$= \mathbf{Transpose}(B'[x_1, y_1, x_2, y_2], \Pi_{4D}(x_1, x_2, y_1, y_2), x_2, y_1)$

Ensure:

$B = LX$ is distributed the same way as X

We present an algorithm for 3D matrix multiplication [10]–[15] that works efficiently given input matrices distributed

cyclically on a 2D processor grid. The algorithm is well-suited for the purposes of analyzing TRSM algorithms. We define the algorithm using a $p_1 \times \sqrt{p_2} \times p_1 \times \sqrt{p_2}$ processor grid, where $\sqrt{p} = p_1\sqrt{p_2}$ in order to provide well-defined transitions from a distribution on a $\sqrt{p} \times \sqrt{p}$ processor grid to faces of a 3D $p_1 \times p_1 \times p_2$ processor grid. The latter 3D processor grid is being used implicitly in our construction. The algorithm assumes divisibility among p, p_1, p_2 and $\sqrt{p_2}$.

A. Cost Analysis of the 3D Matrix Multiplication Algorithm

We analyze the algorithm with account for constant factors in the key leading order costs. The communication costs incurred at line 2, 5, and 7 correspond to the cost of respective collectives, given in Section II-C1.

The transpose on line 4 always occurs on a square processor grid, and so involves only one send and receive of a block. The transposes on line 3 and line 8 are transposes on 2D grids of $p_1 \times \sqrt{p_2}$ processors, with each processor owning nk/p elements. The cost of these transposes is no greater than an all-to-all among $\sqrt{p}$ processors, which can be done with cost $\mathcal{O}(\alpha \cdot \log(p) + \beta \cdot nk\log(p)/p)$. We consider only the asymptotic cost of this transpose, since it will be low order so long as $p_1 \gg 1$. Based on the above arguments, the cost for MM is given line-by-line in the following table.

Line 2	$\alpha \cdot \log(p_2) + \beta \cdot \frac{n^2}{p_1^2}\mathbb{1}_{p_2}$
Line 3	$\mathcal{O}\Big(\alpha \cdot \log(p) + \beta \cdot \frac{nk\log(p)}{p}\Big)$
Line 5	$\alpha \cdot \log(p_1) + \beta \cdot \frac{nk}{p_1p_2}$
Line 4	$\alpha + \beta \cdot \frac{nk}{p}$
Line 6	$\gamma \cdot \frac{n^2k}{p}$
Line 7	$\alpha \cdot \log(p_1) + (\beta+\gamma) \cdot \frac{nk}{p_1p_2}$
Line 8	$\alpha \cdot l + \beta \cdot \frac{nkl}{p}$

To leading order, the cost of MM is given by

$$T_{\mathrm{MM}}(n,k,p,p_1,p_2) = \beta \cdot \Big(\frac{n^2}{p_1^2}\mathbb{1}_{p_2} + \frac{2nk}{p_1p_2}\Big) + \gamma \cdot \frac{n^2k}{p} + \mathcal{O}\Big(\alpha \cdot \log(p) + \beta \cdot \frac{nk\log(p)}{p}\Big).$$

The last communication cost term (due to the rectangular grid transpose) is only of leading order when $p_1 \approx \log(p)$. A square processor grid is not a good initial/final layout for X and B in this case, and the problem would be addressed by choosing an alternative one. We disregard this issue, because we will use the algorithm only for $n \geq k$.

IV. Recursive TRSM

We provide a recursive TRSM algorithm for solving $LX = B$ using the techniques covered in Section II-C4. Our algorithm works recursively on a $p_r \times p_c$ processor grid. We will define the processor grid to be square ($p_r = p_c$) when $n \geq k$, but rectangular ($p_r < p_c$) when $n < k$. So long as $p < k/n$, we will choose $p_c = (k/n)p_r$. This strategy implies the largest of the matrices L and B will be partitioned initially so each cyclically-selected block is close to square. The algorithm starts by partitioning the processor grid into p_c/p_r square grids, if $p_c > p_r$, replicating the matrix L and computing a subset of kp_r/p_c columns of X on each. Then the algorithm partitions L into $n/2 \times n/2$ blocks recursively, executing subproblems with all processors. At a given threshold, n_0, the algorithm stops recursing, gathers L onto all processors, and computes a subset of columns of X with each processor.

$X = \text{Rec-TRSM}(L, B, \Pi_{2\mathrm{D}}, n, k, p_r, p_c, n_0)$

Require:
L is a lower triangular $n \times n$ matrix, distributed on $p_r \times p_c$ in a cyclic layout, so $\Pi_{2\mathrm{D}}(x,y)$ owns $L[x,y]$ of size $\frac{n}{p_r} \times \frac{n}{p_c}$ such that $L[x,y](i,j) = L(ip_r + x, jp_c + y)$.
B is a dense $n \times k$ matrix is distributed cyclically so that $\Pi_{2\mathrm{D}}(x,y)$ owns $X[x,y]$ of size $\frac{n}{p_r} \times \frac{k}{p_c}$

1: **if** $p_r = qp_c$ and $q > 1$ **then**
2: Define $p_r \times p_r \times q$ processor grid $\Pi_{3\mathrm{D}}$, such that $\Pi_{3\mathrm{D}}(x,y,z) = \Pi_{2\mathrm{D}}(x_1, y + p_r z)$ owns blocks $L[x,y,z]$, and $B[x,y,z]$
3: $L'[x,y] = \textbf{Allgather}(L[x,y,\circ], \Pi_{3\mathrm{D}}(x,y,\circ))$
4: $X[\circ,\circ,z] = \textbf{Rec-TRSM}(L'[\circ,\circ], B[\circ,\circ,z], \Pi_{2\mathrm{D}}(\circ,\circ,z), n, k/q, p_r, p_r, n_0)$
5: **else if** $n \leq n_0$ or $p_r = p_c = 1$ **then**
6: $L = \textbf{Allgather}(L[\circ,\circ], \Pi_{2\mathrm{D}}(\circ,\circ))$
7: $B[x + yp_r] = \textbf{AllToAll}(B[\circ,y], \Pi_{2\mathrm{D}}(\circ,y))$
8: $\Pi_{2\mathrm{D}}(x,y) : X[x + yp_r] = L^{-1}B[x + yp_r]$
9: $X[x,y] = \textbf{AllToAll}(B[x + \circ p_r], \Pi_{2\mathrm{D}}(\circ,y))$
10: **else**
11: Partition $L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$ so $L_{ij} \in \mathbb{R}^{\frac{n}{2} \times \frac{n}{2}}$
12: Partition $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$, so $B_i, X_i \in \mathbb{R}^{\frac{n}{2} \times k}$
13: $X_1 = \textbf{Rec-TRSM}(L_{11}, B_1, \Pi_{2\mathrm{D}}, n/2, k, p, p_r, p_c, n_0)$.
14: $B_2' = B_2 - \textbf{MM}(L_{21}, X_1, \Pi_{2\mathrm{D}}, n/2, k, p, p^{1/3}(n/k)^{1/3}, p^{1/3}(n/k)^{2/3})$.
15: $X_2 = \textbf{Rec-TRSM}(L_{22}, B_2', \Pi_{2\mathrm{D}}, n/2, k, p, p_r, p_c, n_0)$.
16: **end if**

Ensure:
$X = L^{-1}B$ is distributed on $\Pi_{2\mathrm{D}}$ in the same way as B

A. Cost Analysis of the Recursive Algorithm

We select $p_c = \max(\sqrt{p}, \min(p, \sqrt{pk/n}))$ and $p_r = p/p_c = \min(\sqrt{p}, \max(1, \sqrt{pn/k}))$. The cost of the allgather on line 3 is

$$T_{\mathrm{part-cols}}(n, p_r) = \mathcal{O}\Big(\beta \cdot \frac{n^2}{p_r^2} + \alpha \cdot \log(p)\Big),$$

since each $L'[x,y] \in \mathbb{R}^{n/p_r \times n/p_r}$ and is lower triangular. Once we have a square processor grid, we partition L, yielding the recurrence,

$$T_{\mathrm{RT}}(n,k,p,n_0) = T_{\mathrm{MM}}\Big(n/2, k, p, p^{1/3}\big(\frac{n}{k}\big)^{1/3}, p^{1/3}\big(\frac{n}{k}\big)^{2/3}\Big) + 2T_{\mathrm{RT}}(n/2,k,p,n_0).$$

We now derive the cost of the algorithm for different relations between n, k, and p, as in the expression for T_{MM}.

One large dimension: When $n < k/p$, we have $p_r = 1$ and $p_c = p$ and the first allgather will be the only communication, therefore,

$$T_{\mathrm{RT1D}}(n,k,p) = \mathcal{O}\bigg(\alpha\cdot\log(p) + \beta\cdot n^2 + \gamma\cdot\frac{n^2k}{p}\bigg).$$

Two large dimensions: When $n > k\sqrt{p}$, we will select $p_r = p_c = \sqrt{p}$ and the column partitioning of B is not performed. In this case, the MM algorithm will always have $p_2 = 1$ (it will be 2D). For sufficiently large k, this leads us to the recurrence,

$$T_{\mathrm{RT2D}}(n,k,p,n_0) = T_{\mathrm{MM2D}}(n/2,k,p) + 2T_{\mathrm{RT2D}}(n/2,k,p,n_0),$$

where $T_{\mathrm{MM2D}}(n,k,p) = \mathcal{O}\left(\alpha\cdot\log(p) + \beta\cdot\frac{nk}{\sqrt{p}} + \gamma\cdot\frac{n^2k}{p}\right)$. The bandwidth cost stays the same at every recursive level, while the computation cost decreases by a factor of 2. At the base case, we incur the cost $T_{\mathrm{RTBC}}(n_0,k,p) =$

$$\mathcal{O}\bigg(\alpha\cdot\log(p) + \beta\cdot\Big(n_0^2 + \frac{n_0k\log(p)}{p}\Big) + \gamma\cdot\frac{n_0^2k}{p}\bigg)$$

We select $n_0 = \max(\sqrt{p}, n\log(p)/\sqrt{p})$, so $n/n_0 \leq \sqrt{p}/\log(p)$, which results in the overall cost,

$$T_{\mathrm{RT2D}}(n,k,p) = \mathcal{O}\bigg(\alpha\cdot\sqrt{p} + \beta\cdot\frac{nk\log(p)}{\sqrt{p}} + \gamma\cdot\frac{n^2k}{p}\bigg).$$

The bandwidth cost above is suboptimal by a factor of $\mathcal{O}(\log(p))$. The overhead is due to the recursive algorithm re-broadcasting some of the same elements of L at every recursive level. We use an iterative approach for our subsequent TRSM algorithm to avoid this redundant communication.

Three large dimensions:

When $k/p < n < k/\sqrt{p}$, the algorithm partitions the columns of B initially then recursively partitions L. In particular, we select $p_c = \max(\sqrt{p}, \sqrt{pk/n})$ and $p_r = p/p_c = \min(\sqrt{p}, \sqrt{pn/k})$, so the first step partitions a rectangular processor grid into $\max(1, k/n)$ fewer processor grids. After the first step, which partitions B, we have independent subproblems with p_r processors. We now start recursively partitioning L, yielding the cost recurrence,

$$\begin{aligned}T_{\mathrm{RT3D}}(n,k,p_r^2,n_0) &= T_{\mathrm{MM3D}}(n/2,k,p_r^2)\\ &+ T_{\mathrm{part-cols}}(n,p)\mathbb{1}_{k/n} + 2T_{\mathrm{RT3D}}(n/2,k,p_r^2,n_0).\end{aligned}$$

Above, we always employ the MM algorithm in the 3D regime by selecting $p_1 = p_r^{2/3}(n/l)^{1/3}$ and $p_2 = p_r^{2/3}(n/l)^{2/3}$ where $l = kp_r/p_c$. In this case, T_{MM} reduces to $T_{\mathrm{MM3D}}(n,k,p) =$

$$\mathcal{O}\bigg(\alpha\cdot\log(p) + \beta\cdot\Big(\Big(\frac{n^2k}{p}\Big)^{2/3} + \frac{nk\log(p)}{p}\Big) + \gamma\cdot\frac{n^2k}{p}\bigg)$$

This gives to the cost recurrence, $T_{\mathrm{RT3D}}(n,k,p_r^2,n_0) =$

$$\begin{aligned}&\mathcal{O}\bigg(\alpha\cdot\log(p) + \beta\cdot\Big(\Big(\frac{n^2k}{p_r^2}\Big)^{2/3} + \frac{n^2}{p_r^2}\mathbb{1}_{\frac{k}{n}}\\ &+ \frac{nk\log(p)}{p_r^2}\Big)\bigg) + \gamma\cdot\frac{n^2k}{p_r^2}\bigg) + 2T_{\mathrm{RT3D}}(\frac{n}{2},k,p_r^2,n_0),\end{aligned}$$

where we can see that $\frac{nk\log(p)}{p_r^2} = \mathcal{O}((n^2k/p_r^2)^{2/3})$, since the initial partitioning will give $n \geq k$. It is also easy to see that $\frac{n^2}{p_r^2}\mathbb{1}_{\frac{k}{n}} = \mathcal{O}((n^2k/p_r^2)^{2/3})$. With these simplifications,

$$\begin{aligned}T_{\mathrm{RT3D}}(n,k,p_r^2,n_0) &= \mathcal{O}\bigg(\alpha\cdot\log(p) + \beta\cdot\Big(\frac{n^2k}{p_r^2}\Big)^{2/3}\\ &+ \gamma\cdot\frac{n^2k}{p_r^2}\bigg) + 2T_{\mathrm{RT3D}}(\frac{n}{2},k,p_r^2,n_0).\end{aligned}$$

We observe that the bandwidth cost $\mathcal{O}((n^2k/p_r^2)^{2/3})$ decreases by a factor of $2^{1/3}$ at every recursive level, and the computation cost by a factor of 2. The base-case cost will be $T_{\mathrm{RTBC}}(n_0,k,p_r^2)$. We select $n_0 = n^{1/3}\left(\frac{k}{p_r^2}\right)^{2/3}$, giving a total cost over all base cases of $\frac{n}{n_0}T_{\mathrm{RTBC}}(n_0,k,p_r^2) =$

$$\begin{aligned}&\mathcal{O}\bigg(\alpha\cdot\frac{n}{n_0}\log(p) + \beta\cdot\Big(nn_0 + \frac{nk\log(p)}{p_r^2}\Big) + \gamma\cdot\frac{nn_0k}{p_r^2}\bigg)\\ &= \mathcal{O}\bigg(\alpha\cdot\Big(\frac{np_r^2}{k}\Big)^{2/3}\log(p) + \beta\cdot\Big(\frac{n^2k}{p_r^2}\Big)^{2/3} + \gamma\cdot\frac{n^{4/3}k^{5/3}}{p_r^{10/3}}\bigg).\end{aligned}$$

Therefore, the overall cost incurred on each square processor grid is $T_{\mathrm{RT3D}}(n,k,p_r^2) =$

$$\mathcal{O}\bigg(\alpha\cdot\Big(\frac{np_r^2}{k}\Big)^{2/3}\log(p) + \beta\cdot\Big(\frac{n^2k}{p_r^2}\Big)^{2/3} + \gamma\cdot\frac{n^2k}{p_r^2}\bigg).$$

When $k \leq n$, we do not have a partitioning step and $p_r^2 = p$. Otherwise, we have $p_r^2 = np/k$ obtain the cost $T_{\mathrm{RT3D}}(n,n,np/k) =$

$$\mathcal{O}\bigg(\alpha\cdot\Big(\frac{np}{k}\Big)^{2/3}\log(p) + \beta\cdot\Big(\frac{n^2k}{p}\Big)^{2/3} + \gamma\cdot\frac{n^2k}{p}\bigg),$$

which is the same as for the case $k \leq n$. For $n = k$, the 3D costs obtained above are the same as the most efficient algorithms for $n \times n$ LU factorization. In the subsequent sections, we show that a lower synchronization cost is achievable via selective use of triangular matrix inversion.

V. Triangular Inversion

In this section, we derive the cost of inverting a lower triangular matrix L of size $n \times n$ with p processors. Since the input matrix is square, the dimensions of the processor grid Π should be identical in two dimensions leaving us with $\dim(\Pi) = p_1 \times p_1 \times p_2$, where $p = p_1^2p_2$. We assume the initial matrix to be cyclically distributed on the subgrid $\Pi(\circ,\circ,1)$.

A. Algorithmic Approach

In [23], a recursive method for inverting triangular matrices was presented. A similar method for full inversion was presented in [24]. When applied to a triangular matrix, those methods coincide. The method uses the triangular structure of the initial matrix to calculate the inverse by subdividing the problem into two recursive matrix inversion calls, which can be executed concurrently and then uses two matrix multiplications to complete the inversion.

Since the subproblems are independent, we want to split the processor grid such that two distinct sets of processors work on either subproblem. We chose the base case condition to be that the grid is one-dimensional in the dimension of p_1 and we do redundant base case calculations in this subgrid. For this section, we consider $p_2 \geq p_1$, a constraint that we will fulfill anytime the method is called.

$L^{-1} = \text{RecTriInv}(L, \Pi, p, p_1, p_2)$

Require:
The processor grid Π has dimensions $\sqrt{p} \times \sqrt{p}$
L is a lower triangular $n \times n$ matrix, distributed on Π in a cyclic layout, so $\Pi(x, y)$ owns $L[x, y]$ of size $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ such that $L[x, y](i, j) = L(i\sqrt{p} + x, j\sqrt{p} + y)$.

if $p_1 = 1$ **then**
  AllToAll $(L[x, \circ], \Pi(x, \circ))$
  $L^{-1} =$ sequential inversion(L)
else
  Subdivide L into $n/2 \times n/2$ blocks,
  $L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$
  Subdivide the processor grid $\Pi = [\Pi_1, \Pi_2]$ such that $\dim(\Pi_1) = \dim(\Pi_2) = (\sqrt{p/2} \times \sqrt{p/2})$
  Redistribute $(L_{11}, \Pi \to \Pi_1)$
  Redistribute $(L_{22}, \Pi \to \Pi_2)$
  $L_{11}^{-1} =$ Rec-Tri-Inv$(L_{11}, \Pi_1, p, p_1/2^{2/3}, p_2/2^{2/3})$
  $L_{22}^{-1} =$ Rec-Tri-Inv$(L_{22}, \Pi_2, p, p_1/2^{2/3}, p_2/2^{2/3})$
  $L_{21}'^{-1} = -$MM$(L_{22}^{-1}, L_{21}, \Pi, n, n, p, p_1, p_2)$
  $L_{21}^{-1} =$ MM$(L_{21}'^{-1}, L_{11}^{-1}, \Pi, n, n, p, p_1, p_2)$
  Assemble L^{-1} from the $n/2 \times n/2$ blocks,
  $L^{-1} = \begin{bmatrix} L_{11}^{-1} & 0 \\ L_{21}^{-1} & L_{22}^{-1} \end{bmatrix}$
end if

Ensure:
$LL^{-1} = \mathbf{1}$ where L^{-1} is distributed the same way as L

B. Total Cost of Triangular Inversion

This recursive approach of inverting a matrix has total cost,

$$T_{\text{RecTriInv}}(n, p_1, p_2) = 2T_{\text{MM}}(n/2, n/2, p_1, p_2) + T_{\text{RecTriInv}}(n/2, p_1/2^{1/3}, p_2/2^{1/3}) + T_{\text{redistr}}(n/2, p_1, p_2),$$

with a base case cost of

$$T_{\text{RecTriInv}}(n_0, 1, p_2) = \alpha \cdot 2 \log\left(\frac{p_2}{p_1}\right) + \beta \cdot 2n_0^2 + \gamma \cdot n_0^3.$$

The base case size will be $n_0 = \frac{n}{p_1^{3/2}}$ and therefore neither of the terms is of leading order. We observe that the bandwidth cost of the matrix multiplication $\mathcal{O}((n^3/p_1^2)^{2/3})$ decreases by a factor of $2^{4/9}$ at every recursive level, and the computation cost by a factor of 2. The redistribution process requires moving the matrices from a cyclic processor grid to a smaller cyclic processor grid, with the block each processor owns having a factor of $2^{1/3}$ more rows and columns. This redistribution is effectively an all-to-all between a larger and a smaller set of processors. We can get a concrete bound on the cost, by first performing an all-to-all to transition to a blocked layout (each processor owns contiguous blocks of the matrix). Then we can transition to a blocked layout on the smaller processor grid by scattering each block to at most 4 processors. Finally, we can perform an all-to-all on the smaller processor grid to transition from the blocked layout back to a cyclic one. The overall cost of these steps is $O(\alpha \cdot \log(p) + \beta \cdot n_0^2 \log(p)/p)$ This redistribution bandwidth cost is dominated by the cost of the matrix multiplication.

The total cost for the recursive inversion is

$$T_{\text{RecTriInv}}(n, p_1, p_2) = \beta \cdot \frac{2^{1/3}}{2^{1/3} - 1}\left(\frac{n^2}{8p_1^2} + \frac{n^2}{2p_1p_2}\right) + \gamma \cdot \frac{2^{1/3}}{2^{1/3} - 1}\frac{1}{8}\frac{n^3}{p} + \mathcal{O}\left(\alpha \log^2 p\right).$$

In contrast to LU factorization and our recursive TRSM algorithm, the synchronization cost is logarithmic rather than polynomial in p.

VI. Iterative Triangular Solver

In this section, we present our main contribution, a 3D TRSM algorithm that uses inversion of diagonal blocks to achieve a lower synchronization cost. By precomputing the inversions, we replace the latency-dominated small TRSMs with more parallel matrix multiplications.

A. Block-Diagonal Triangular Inversion

In order to lower the synchronization cost of TRSM, first we invert a set of triangular blocks along the diagonal of the matrix, each with a distinct subset of processors. We split Π into $\frac{n}{n_0}$ subgirds of dimensions $r_1 \times r_1 \times r_2$, where $r_1^2 r_2 = p\frac{n_0}{n}$. To have the proper layout for the inversion, a transition from the original, cyclic layout on a subgrid to the grid as described in Section III has to happen. Afterwards, all the inversions can be done in parallel. To support our inversion, we must have $r_2 > r_1$ and $n_0 \geq \sqrt{r_1^2 r_2}$. The precise choices of r_1 and r_2 are given in the algorithm and will be discussed in Section VII.

B. Triangular Solve using Partial Inversion

Initially, we want L to be distributed on the top level of the three dimensional grid Π in a cyclic layout such that each processor $\Pi(x, y, 1)$ owns $L\left(y : p_1 : n, x : p_1 : n\right)$. Also, we set the right hand side to be distributed on one level of the grid with a blocked layout with a physical block size of $b \times \frac{k}{p_2}$ such that each processor $\Pi(x, 1, z)$ owns $B\left(x : p_1 : \frac{n}{b}, zk/p_2 : (z+1)k/p_2\right)$.

$\tilde{L}$ = Diagonal-Inverter$(L, \Pi, n, p_1, p_2, n_0)$

Require:
The processor grid Π has dimensions $p_1 \times p_1 \times p_2$
L is a lower triangular $n \times n$ matrix distributed cyclically on Π such that processor $\Pi(x, y, 1))$ owns $L[x, y]$ a lower triangular $\frac{n}{p_1} \times \frac{n}{p_1}$ matrix such that $L[x, y](i, j) = L(ip_1 + x, jp_1 + y)$.

1: Define $q = \frac{pn_0}{n} \quad r = \frac{n}{n_0}$
2: Define $r_1 = \left(\frac{pn_0}{4n}\right)^{1/3}$
3: Define $r_2 = \left(\frac{16pn_0}{n}\right)^{1/3}$
4: Define a $p_1 \times p_1 \times \sqrt{p_2} \times \sqrt{p_2}$ processor grid Π_{4D}, such that $\Pi_{4D}(x_1, x_2, y_1, y_2) = \Pi(x_1, x_2, y_1 + \sqrt{p_2}y_2)$.
and $\Pi_{4D}(x_1, x_2, y_1, y_2)$ owns blocks $L[x_1, x_2, y_1, y_2]$
5: Define a block diagonal matrix $L_D[x_1, x_2, y_1, y_2]$ such that $L_D[x_1, x_2, y_1, y_2][b]$ denotes the block $L[x_1, x_2, y_1, y_2](bn_0 : (b+1)n_0, bn_0 : (b+1)n_0)$
6: **Scatter** $(L_D[\circ, \circ, y_1, y_2][\circ], \Pi_{4D}(x_1, x_2, 1, 1), \Pi_{4D}(x_1, x_2, \circ, \circ))$
7: Define a $\sqrt{p} \times \sqrt{p}$ processor grid Π_{2D}, such that $\Pi_{2D}(x_1 + p_1 y_1, x_2 + p_2 y_2) = \Pi_{4D}(x_1, x_2, y_1, y_2)$.
8: Define a $\sqrt{q} \times \sqrt{q} \times \sqrt{r} \times \sqrt{r}$ processor grid Π^I_{4D}, such that $\Pi^I_{4D}(u_1, u_2, v_1, v_2) = \Pi_{2D}(u_1 + \sqrt{q}v_1, u_2 + \sqrt{q}v_2)$ owns blocks $L_D[u_1, u_2, v_1, v_2]$.
9: **AllToAll** $\left(L_D[u_1, u_2, v_1, v_2][\circ], \Pi^I_{4D}(u_1, u_2, \circ, \circ)\right)$
10: **For** $i = 0 : \sqrt{\frac{n}{n_0}} - 1$ **do in parallel**
11: **For** $j = 0 : \sqrt{\frac{n}{n_0}} - 1$ **do in parallel**
12: Define $b = \left(i + \sqrt{\frac{n}{n_0}}j\right)$
13: $\tilde{L}_D[\circ, \circ, i, j][b] =$ **RecTriInv** $(L_D[\circ, \circ, i, j][b], \Pi^I_{4D}[\circ, \circ, i, j], q, r_1, r_2))$
14: **end for**
15: **end for**
16: **AllToAll** $\left(\tilde{L}_D[u_1, u_2, v_1, v_2][\circ], \Pi^I_{4D}(u_1, u_2, \circ, \circ)\right)$
17: **Gather** $\left(\tilde{L}_D[\circ, \circ, y_1, y_2][\circ], \Pi_{4D}(x_1, x_2, \circ, \circ), \Pi_{4D}(x_1, x_2, 1, 1)\right)$

Ensure:
$\tilde{L}_D L_D = \mathbf{1} \quad \forall i$ where L and $\tilde{L}$ are partitioned the same way

Additionally, each processor has memory of the same size as its part of B allocated for an update-matrix denoted as $\overline{B_j}, \quad j \in [1, p_1]$, where also each processor $\Pi(x, y, z)$ owns $\overline{B_y}\left(x : p_1 : \frac{n}{b}, zk/p_2 : (z+1)k/p_2\right)$. The algorithm itself consists of two parts: first 'inversion', we invert all the base-cases on the diagonal in parallel as described in the algorithm above and, second 'solve', we do the updates and calculate the solution to TRSM.

VII. Cost Analysis of the Iterative TRSM

In this section we will derive a performance model for the algorithm presented in Section VI. The total cost of the algorithm is put together from the cost of its three subroutines:

$$T_{\text{It-Inv-TRSM}}(n, k, n_0, p_1, p_2) = T_{\text{Inv}}(n, p_1, p_2) + T_{\text{Upd}}(n, k, n_0, p_1, p_2) + T_{\text{Solve}}(n, k, n_0, p_1, p_2).$$

Above the cost denoted by inversion is the part of the algorithm that inverts the blocks (Algorithm Diagonal-Inverter). The solve part is in lines 4-5, and the update in lines 7-8.

X = It-Inv-TRSM$(L, B, \Pi, n, k, p_1, p_2, r_1, r_2)$

Require:
The processor grid Π has dimensions $p_1 \times p_1 \times p_2$
L is a lower triangular $n \times n$ matrix is distributed on Π such that $\Pi(x, y, 1)$ owns $L[x, y]$ of size $n/p_1 \times n/p_1$ such that $L[x, y](i, j) = L(ip_1 + x, jp_1 + y)$
B is a dense $n \times k$ matrix is distributed such that $\Pi(x, 1, z)$ owns $B[x, z]$of size$n/p_1 \times k/p_2$, such that $B[x, z](i, j) = L(ip_1 + x, zk/p_2 + j)$
Define blocks $S_i = in_0 : (i+1)n_0$ and $T_i = in_0 : n$

1: $\tilde{L} =$ **Diagonal-Inverter**$(L, \Pi, n, p_1, p_2, n_0)$
2: **Bcast** $(B[x, z](S_0(x), \circ), \Pi(x, 1, z), \Pi(x, \circ, z))$
3: **for** $i = 0 : \frac{n}{n_0} - 1$ **do**
4: $\Pi(x, y, z) : X[y, z](S_i, \circ) = \tilde{L}[y, x](S_i, S_i) \cdot B[x, z](S_i, \circ)$
5: $X[y, z](S_i, \circ) =$ **Allreduce** $(X[y, z](S_i, \circ), \Pi(\circ, y, z))$
6: **Bcast** $\left(\tilde{L}[x, y](T_{i+1}, S_i), \Pi(x, y, 1), \Pi(x, y, \circ)\right)$
7: $\Pi(x, y, z) : \overline{B_y}[x, z](T_{i+1}, \circ) += \tilde{L}[x, y](T_{i+1}, S_i) \cdot X[y, z](S_i, \circ)$
8: $\overline{B_0}[x, z](S_{i+1}, \circ) =$ **Allreduce**$(\overline{B_\circ}[x, z](S_{i+1}, \circ), \Pi(x, \circ, z))$
9: $\Pi(x, y, z) : B[x, z](S_{i+1}, \circ) = B[x, z](S_{i+1}, \circ) - \overline{B_0}[x, z](S_{i+1}, \circ)$
10: **end for**

Ensure:
$B = LX$ where X is distributed the same way as B

A. Inversion Cost

We invert the $\frac{n}{n_0}$ submatrices of size $n_0 \times n_0$ along the diagonal with distinct processor grids. The size of the processor grids involved is $r_1 \times r_1 \times r_2$. The choices of r_1 and r_2 are made such that the bandwidth cost of the inversion is minimal. Additionally we have to account for the cost that arises from communicating the submatrices to the proper subgrids. This happens in lines 6, 9, 16, and 17 of the Algorithm Diagonal-Inverter. The respective costs are summed up the the following table.

Line 6	$\alpha \cdot \log(p_2) + \beta \cdot \frac{nn_0}{2p_1^2}$
Line 9	$\mathcal{O}\left(\alpha \cdot \log(p) + \beta \cdot \frac{nn_0 \log p}{2p}\right)$
Line 16	$\mathcal{O}\left(\alpha \cdot \log(p) + \beta \cdot \frac{nn_0 \log p}{2p}\right)$
Line 17	$\alpha \cdot \log(p_2) + \beta \cdot \frac{nn_0}{2p_1^2}$

These cost are never of leading order compared to the costs that arise form the triangular inversion. With the derivations done in Section V, we get the following costs for inversion:

Latency Cost: The total latency cost of inversion is

$$S_{\text{Inv}}(p) = \mathcal{O}\left(\alpha \log^2 p\right).$$

Bandwidth Cost: In order to minimize the bandwidth cost of triangular inversion, we choose a grid splitting to achieve closest to ideal ratios for the subgrids processor layout r_1 and r_2. This ratio is achieved when $r_2 = 4r_1$. The choices for r_1 and r_2 are therefore,

$$r_1 = \left(\frac{pn_0}{4n}\right)^{1/3} \quad \text{and} \quad r_2 = \left(\frac{16pn_0}{n}\right)^{1/3}.$$

With this grid slicing, we get $\frac{n}{n_0}$ different sub-grids of dimensions $r_1 \times r_1 \times r_2$. This setup leads to a cost for inverting $\frac{n}{n_0}$ submatrices of

$$W_{\text{Inv}}(n_0, r_1, r_2) = \frac{2^{1/3}}{2^{1/3}-1}\left(\frac{n_0^2}{8r_1^2} + \frac{n_0^2}{2r_1r_2}\right).$$

Flop Cost: The flop cost of the inversion part is

$$F_{\text{Inv}}(n_0, p_1, p_2) = \frac{1}{8}\frac{nn_0^2}{p_1^2p_2}.$$

B. Solve Cost

The complete solve cost can be derived by

$$T_{\text{Solve}}(n, n_0, k, p, p_1, p_2) = \frac{n}{n_0} T_{\text{MM}}(n_0, k, p, p_1, p_2)$$

Latency Cost: The latency cost of the solve part is

$$S_{\text{Solve}}(n, n_0, p) = \mathcal{O}\left(\frac{n}{n_0}\log p\right).$$

Bandwidth Cost: The cost of the solve is one call to triangular matrix multiplication for each base case. The synchronization cost is again dominated by the $\frac{n}{n_0}$ cases. The cost for these sums has been presented in Section III. Including these, we obtain a total cost of

$$\begin{aligned} W_{\text{Solve}}(n, k, n_0, p, p_1, p_2) &= \frac{n}{n_0} \cdot W_{\text{MM}}(n_0, k, p, p_1, p_2) \\ &= \frac{n}{n_0} \cdot \left[\left(\frac{n_0^2}{p_1^2}\right)\mathbb{1}_{p_2} + 4\left(\frac{n_0k}{p_1p_2}\right)\mathbb{1}_{p_1}\right]. \end{aligned}$$

Flop Cost: The flop cost of the solve part is

$$F_{\text{Solve}}(n, k, n_0, p_1, p_2) = \frac{n}{n_0}\left(\frac{n_0^2k}{p_1^2p_2}\right).$$

C. Update Cost

The complete solve cost can be derived by

$$T_{\text{Upd}}(n, k, n_0, p, p_1, p_2) = \sum_{i=1}^{n/n_0-1} T_{\text{MM}}(n - in_0, n_0, k, p, p_1, p_2).$$

Latency Cost: The update latency cost is

$$S_{\text{Upd}}(n, n_0, p) = \mathcal{O}\left(\frac{n-n_0}{n_0}\log p\right).$$

Bandwidth Cost: The cost of doing all the updates as described in the algorithm in Section VI (Lines 5-9) is the cost of both the allreductions and the broadcast,

$$\begin{aligned} W_{\text{Upd}}(n, k, n_0, p_1, p_2) = \\ \sum_{i=1}^{n/n_0-1}\left[W_{\text{bcast}}\left(\frac{nn_0 - in_0}{p_1^2}, p_2\right) + \right. \\ \left. W_{\text{allreduction}}\left(\frac{n_0k}{p_1p_2}, p_1\right) + W_{\text{allreduction}}\left(\frac{n_0k}{p_1p_2}, p_1\right)\right]. \end{aligned}$$

This yields to a total cost of $W_{\text{Upd}}(n, k, n_0, p_1, p_2) =$

$$\frac{n-n_0}{n_0}\left[4\frac{nn_0-n}{p_1^2}\mathbb{1}_{p_2} + 4\frac{n_0k}{p_1p_2}\mathbb{1}_{p_1}\right].$$

Flop Cost: The update flop cost is

$$F_{\text{Upd}}(n, k, n_0, p_1, p_2) = \frac{n-n_0}{n_0}\left(\frac{knn_0}{p_1^2p_2}\right).$$

D. Total Cost

The total cost of the algorithm is the sum of its three parts and leaves a lot of tuning room as with a choice of $p_1 = 1$, $p_2 = 1$ or $n_0 = n$ one is able to eliminate certain terms.

Latency Cost: The total latency cost of the algorithm is a sum of the previous parts,

$$\begin{aligned} S_{\text{It-Inv-TRSM}}(p_1, p_2, r_1, b) = S_{\text{Upd}}(n, n_0, p_1, p_2) + \\ S_{\text{Solve}}(n, n_0, p_1, p_2) + S_{\text{Inv}}(p_1, p_2, r_1, b) \\ = \mathcal{O}\left(\alpha\left(\frac{n}{n_0}\log p + \log^2 p\right)\right). \end{aligned}$$

Bandwidth Cost: The total bandwidth cost for the TRSM algorithm is, by abbreviating $\nu = \frac{2^{1/3}}{2^{1/3}-1}$,

$$\begin{aligned} W_{\text{It-Inv-TRSM}}(n, k, n_0, p_1, p_2, u, v, b) = \\ W_{\text{Upd}}(n, k, n_0, p_1, p_2) + W_{\text{Solve}}(n, k, n_0, p_1, p_2) \\ + W_{\text{Inv}}(n, b, p_1, p_2, u, v) \\ = \frac{n}{n_0} \cdot \left[\left(\frac{n_0^2}{p_1^2}\right)\mathbb{1}_{p_2} + 4\left(\frac{n_0k}{p_1p_2}\right)\mathbb{1}_{p_1}\right] \\ + \frac{n-n_0}{n_0}\left[4\frac{nn_0-n}{p_1^2}\mathbb{1}_{p_2} + 4\frac{n_0k}{p_1p_2}\mathbb{1}_{p_1}\right] + \nu\left(\frac{n_0^2}{8r_1^2} + \frac{n_0^2}{2r_1r_2}\right). \end{aligned}$$

Flop Cost: Lastly, the combined total flop cost is

$$\begin{aligned} F_{\text{It-Inv-TRSM}}(n, k, n_0, p_1, p_2) = F_{\text{Upd}}(n, n_0, p_1, p_2) \\ + F_{\text{Solve}}(n, n_0, p_1, p_2) + F_{\text{Inv}}(n_0, u, v, p_1, p_2) = \frac{n^2k}{p_1^2p_2} + \frac{n_0^2n}{p_1^2p_2}. \end{aligned}$$

VIII. Parameter Tuning

In this section, we give asymptotically optimal tuning parameters for different relative matrix sizes to optimize performance. We only focus on asymptotic parameters as there is a trade off between the constant factors on the bandwidth and latency costs. The exact choice is therefore machine

dependent and should be determined experimentally. The initial grid layout is dependent on the relative matrix sizes of L and B since the update part of the algorithm is one of the dominating terms in any case where there is an update to be made and determines the case where is is infeasible. The different layouts are shown in Figure 1.

In the case where $n < \frac{4k}{p}$, the processor grid layout is one-dimensional. The optimal parameters are given in the following table.

$p_1 = 1$	$r_1 = \mathcal{O}\left((p)^{1/3}\right)$	$n_0 = n$
$p_2 = p$	$r_2 = \mathcal{O}\left((p)^{1/3}\right)$	

Using this set of parameters will yield to a total cost of

$$T_{\mathrm{IT1D}}(n,k,p) = \mathcal{O}\left(\alpha \cdot \left(\log^2 p + \log p\right) + \beta \cdot n^2 + \gamma \cdot \frac{n^2 k}{p}\right).$$

Comparing these costs to the costs of T_{RT1D} obtained in Section IV-A, we can see that we are within the asymptotic bounds of the original algorithm in bandwidth and flop cost, but pay an extra factor of $\log p$ latency, since the inversion, if performed on a 3D grid, requires $\log^2 p$ steps. But since the inversion is the least significant part of the routine in 1 large dimension, no gain was to be expected in this case.

In the case where $n > 4k\sqrt{p}$, the processor grid layout is two-dimensional. The optimal parameters are given in the following table.

$p_1 = \sqrt{p}$	$p_2 = 1$	$n_0 = \mathcal{O}\left(\left(nk^3p^{1/2}\right)^{1/4}\right)$
$r_1 = \mathcal{O}\left(\left(\frac{k}{n}\right)^{1/4} p^{3/8}\right)$	$r_2 = \mathcal{O}\left(\left(\frac{k}{n}\right)^{1/4} p^{3/8}\right)$	

Using this set of parameters will yield to a total cost of

$$T_{\mathrm{IT2D}}(n,k,p) = \mathcal{O}\left(\alpha\left(\log^2 p + \left(\frac{n}{k}\right)^{3/4}\frac{1}{p^{1/8}}\log p\right) + \beta\left(\frac{nk}{\sqrt{p}}\right) + \gamma\left(\frac{n^2k}{\sqrt{p}}\right)\right).$$

Comparing these costs to the cost of T_{RT2D} obtained in Section IV-A, we can see that we are asymptotically more efficient in terms of latency by a factor of at least $\frac{p^{1/4}}{\log p}$ as well as in bandwidth by a factor of $\log p$ while having the same flop cost asymptotically. This is a significant gain and especially important as the occurrence of fewer right hand sides $k < n$ is high.

In the case where $\frac{4k}{p} \leq n \leq 4k\sqrt{p}$, the processor grid layout is three-dimensional. The optimal parameters are given in the following table.

$p_1 = \left(\frac{pn}{4k}\right)^{1/3}$	$p_2 = \left(\frac{\sqrt{p}4k}{n}\right)^{2/3}$	$n_0 = \mathcal{O}\left(\min\left(\sqrt{nk}, n\right)\right)$
$r_1 = \mathcal{O}\left(\left(\min\left[\frac{p\sqrt{nk}}{n}, p\right]\right)^{1/3}\right)$	$r_2 = \mathcal{O}\left(\left(\min\left[\frac{p\sqrt{nk}}{n}, p\right]\right)^{1/3}\right)$	

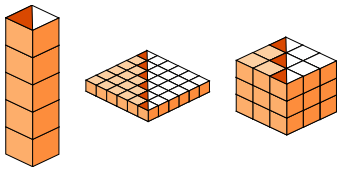

Figure 1. One-, two-, and three-dimensional layout dependent on relative matrix sizes. Inverted blocks of the matrix in dark and input- and output of the right hand side on the left and right size of the cuboid.

Using this set of parameters will yield to a total cost of

$$T_{\mathrm{IT3D}}(n,k,p) = \mathcal{O}\left(\alpha\left(\log^2 p + \max\left(\sqrt{\frac{n}{k}}, 1\right)\log p\right) + \beta\left(\left(\frac{n^2k}{p}\right)^{2/3}\right) + \gamma\left(\frac{n^2k}{\sqrt{p}}\right)\right).$$

Comparing these costs to the cost of T_{RT3D} obtained in Section IV-A, we can see that we are asymptotically more efficient in terms of latency by a factor of $\left(\frac{n}{k}\right)^{1/6} p^{2/3}$ while being able to keep bandwidth and flop costs asymptotically constant.

IX. Conclusion

	S	W	F
	1 Large Dimension $\left(n < \frac{4k}{p}\right)$		
standard	$\log p$	n^2	$\frac{n^2k}{p}$
new method	$\log^2 p$	n^2	$\frac{n^2k}{p}$
	2 Large Dimensions $\left(n > 4k\sqrt{p}\right)$		
standard	$\sqrt{p}$	$\log p \frac{nk}{\sqrt{p}}$	$\frac{n^2k}{p}$
new method	$\log^2 p + \left(\frac{n}{k}\right)^{3/4}\frac{1}{p^{1/8}}\log p$	$\frac{nk}{\sqrt{p}}$	$\frac{n^2k}{p}$
	3 Large Dimensions $\frac{4k}{p} \leq n \leq 4k\sqrt{p}$		
standard	$\left(\frac{np}{k}\right)^{2/3}\log p$	$\left(\frac{n^2k}{p}\right)^{2/3}$	$\frac{n^2k}{p}$
new method	$\log^2 p + \sqrt{\frac{n}{k}}\log p$	$\left(\frac{n^2k}{p}\right)^{2/3}$	$\frac{2n^2k}{p}$

We present a new method for solving triangular systems for multiple right hand sides. In the above table, we compare to a baseline algorithm adapted from [3] that achieves costs that are as good or better than the state of the art [18], [21], [25]. Our algorithm achieves better theoretical scalability than these alternatives by up to a factor of $\left(\frac{n}{k}\right)^{1/6} p^{2/3}$. For certain matrix dimensions, a decrease of bandwidth cost by a factor of $\log_2 p$ is obtained by use of selective triangular matrix inversion. By only inverting triangular blocks along the diagonal of the initial matrix, we generalize the usual way of TRSM computation and the full matrix inversion approach. Fine-tuning the algorithm based on the relative input sizes as well as the number of processors available leads to a significantly more robust algorithm. The cost

analysis of this new method allows us to give recommendations for asymptotically optimal tuning parameters for a wide variety of possible inputs. The detailed pseudo-code provides a direct path toward a more efficient parallel TRSM implementation.

REFERENCES

[1] F. G. Gustavson, "Recursion leads to automatic variable blocking for dense linear-algebra algorithms," *IBM Journal of Research and Development*, vol. 41, no. 6, pp. 737–755, 1997.

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

[3] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström, "Recursive blocked algorithms and hybrid data structures for dense matrix library software," *SIAM review*, vol. 46, no. 1, pp. 3–45, 2004.

[4] J. J. Du Croz and N. J. Higham, "Stability of methods for matrix inversion," *IMA Journal of Numerical Analysis*, vol. 12, no. 1, pp. 1–19, 1992.

[5] E. Solomonik, E. Carson, N. Knight, and J. Demmel, "Tradeoffs between synchronization, communication, and computation in parallel linear algebra computations," in *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '14. New York, NY, USA: ACM, 2014, pp. 307–318.

[6] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, pp. 1749–1783, 2007.

[7] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[8] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 11, pp. 1143–1156, 1997.

[9] J. L. Träff and A. Ripke, "Optimal broadcast for fully connected processor-node networks," *Journal of Parallel and Distributed Computing*, vol. 68, no. 7, pp. 887–901, 2008.

[10] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM Journal on Computing*, vol. 10, no. 4, pp. 657–675, 1981.

[11] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar, "A three-dimensional approach to parallel matrix multiplication," *IBM J. Res. Dev.*, vol. 39, pp. 575–582, September 1995.

[12] A. Aggarwal, A. K. Chandra, and M. Snir, "Communication complexity of PRAMs," *Theoretical Computer Science*, vol. 71, no. 1, pp. 3 – 28, 1990.

[13] J. Berntsen, "Communication efficient matrix multiplication on hypercubes," *Parallel Computing*, vol. 12, no. 3, pp. 335–342, 1989.

[14] W. F. McColl and A. Tiskin, "Memory-efficient matrix multiplication in the BSP model," *Algorithmica*, vol. 24, pp. 287–297, 1999.

[15] S. L. Johnsson, "Minimizing the communication time for matrix multiplication on multiprocessors," *Parallel Comput.*, vol. 19, pp. 1235–1257, November 1993.

[16] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, "Communication-optimal parallel recursive rectangular matrix multiplication," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013, pp. 261–272.

[17] A. Tiskin, "Bulk-synchronous parallel gaussian elimination," *Journal of Mathematical Sciences*, vol. 108, no. 6, pp. 977–991, 2002.

[18] M. T. Heath and C. H. Romine, "Parallel solution of triangular systems on distributed-memory multiprocessors," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 3, pp. 558–588, 1988.

[19] P. Raghavan, "Efficient parallel sparse triangular solution using selective inversion," *Parallel Processing Letters*, vol. 8, no. 01, pp. 29–40, 1998.

[20] D. Irony and S. Toledo, "Trading replication for communication in parallel distributed-memory dense solvers," *Parallel Processing Letters*, vol. 12, no. 01, pp. 79–94, 2002.

[21] B. Lipshitz, "Communication-avoiding parallel recursive algorithms for matrix multiplication," Master's thesis, University of California, Berkeley, 2013.

[22] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense linear algebra solvers for multicore with GPU accelerators," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–8.

[23] A. Borodin and I. Munro, *The computational complexity of algebraic and numeric problems*. Elsevier Publishing Company, 1975, vol. 1.

[24] S. M. Balle, P. C. Hansen, and N. Higham, "A Strassen-type matrix inversion algorithm," *Advances in Parallel Algorithms*, pp. 22–30, 1994.

[25] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK User's Guide*, J. J. Dongarra, Ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997.