Generalizing Random Butterfly Transforms to Arbitrary Matrix Sizes

NEIL LINDQUIST, The University of Tennessee, Knoxville, TN, USA
PIOTR LUSZCZEK, MIT Lincoln Laboratory and The University of Tennessee, Lexington, MA, USA
JACK DONGARRA, The University of Tennessee, Knoxville, TN, USA

Parker and Lê introduced random butterfly transforms (RBTs) as a preprocessing technique to replace pivoting in dense LU factorization. Unfortunately, their FFT-like recursive structure restricts the dimensions of the matrix. Furthermore, on multinode systems, efficient management of the communication overheads restricts the matrix's distribution even more. To remove these limitations, we have generalized the RBT to arbitrary matrix sizes by truncating the dimensions of each layer in the transform. We expanded Parker's theoretical analysis to generalized RBT, specifically that in exact arithmetic, Gaussian elimination with no pivoting will succeed with probability 1 after transforming a matrix with full-depth RBTs. Furthermore, we experimentally show that these generalized transforms improve performance over Parker's formulation by up to 62% while retaining the ability to replace pivoting. This generalized RBT is available in the SLATE numerical software library.

CCS Concepts: • **Mathematics of computing** → **Solvers**; **Mathematical software performance**; **Computations on matrices**; • **Computing methodologies** → *Distributed algorithms*;

Additional Key Words and Phrases: Gaussian Elimination, Randomization

ACM Reference format:
Neil Lindquist, Piotr Luszczek, and Jack Dongarra. 2024. Generalizing Random Butterfly Transforms to Arbitrary Matrix Sizes. *ACM Trans. Math. Softw.* 50, 4, Article 26 (December 2024), 23 pages.
https://doi.org/10.1145/3699714

This research was funded in part by the National Science Foundation Office of Advanced Cyberinfrastructure under Grant No. 2004541. This research was also supported by the Exascale Computing Project, a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. Finally, this research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Research was sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as presenting the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
Authors' Contact Information: Neil Lindquist (corresponding author), The University of Tennessee, Knoxville, TN, USA; e-mail: nlindquist@acm.org; Piotr Luszczek, MIT Lincoln Laboratory and The University of Tennessee, Lexington, MA, USA; e-mail: luszczek@icl.utk.edu; Jack Dongarra, The University of Tennessee, Knoxville, TN, USA; e-mail: dongarra@icl.utk.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1557-7295/2024/12-ART26
https://doi.org/10.1145/3699714

1 Introduction

Gaussian elimination with partial pivoting (GEPP) is commonly used to solve large, dense systems of linear equations. For numerical stability, the computational elimination is interleaved with the row exchanges to maximize the magnitude of the diagonal elements, which subsequently scale the outer-product updates. However, most pivoting procedures incur significant communication overheads due to the growing performance imbalance between computational capacity and communication bandwidth available on modern supercomputers. Furthermore, pivoting adds data dependencies that limit the available parallelism, not only asymptotically but also in most practical settings. Unfortunately, **Gaussian elimination with no pivoting (GENP)** is numerically unstable for most linear systems originating in scientific applications. An alternative to pivoting is preprocessing the matrix with a random transform before factoring it with GENP; such randomization provides numerical stability by preventing large element growth. A popular choice for this preprocessing is Parker and Lê's **random butterfly transform (RBT)** [Parker and Lê, 1995].

RBT preprocessing is similar to combined left- and right-preconditioning, except the "preconditioned" matrix is explicitly computed so that it can be factored. In other words, for a pair of transforms, $\mathcal{U}^T$ and $\mathcal{V}$, the linear system $Ax = b$ is rewritten as

$$(\mathcal{U}^T A \mathcal{V})(\mathcal{V}^{-1} x) = (\mathcal{U}^T b). \tag{1}$$

(While we use $\mathcal{U}^T$ here, $\mathcal{U}^*$ can also be used in the complex case.) From Equation (1), the algorithm falls out naturally as follows:

(1) Transform the system as $\widetilde{A} \equiv \mathcal{U}^T A \mathcal{V}$ and $\widetilde{b} \equiv \mathcal{U}^T b$.
(2) Solve $\widetilde{A}\widetilde{x} = \widetilde{b}$ without pivoting.
(3) Transform the solution by $\mathcal{V}$ to undo the implicit $\mathcal{V}^{-1}$.

This approach requires fast transforms to avoid adding a significant overhead to the factorization. The RBT has a **fast Fourier transform (FFT)**-like structure which limits the cost of transforming $A \in \mathbb{R}^{n \times n}$ to $\mathcal{O}(n^2 \log_2(n))$. Furthermore, the transform is often truncated to reduce the logarithmic term to a small constant.

Unfortunately, RBTs are limited to matrix sizes that are multiples of 2^d (where d is the depth) due to the FFT-like structure. Furthermore, efficient application in distributed settings requires the matrix distribution to be aligned to the butterfly structure [Baboulin et al., 2014; Lindquist et al., 2020]. Thus, applications must pad their matrices with the identity matrix to fit the transform, requiring more operations in the factorization and a matrix allocation that depends on the transform's depth. However, the preprocessing of linear systems does not depend on the exact numerical properties of the chosen RBT but simply on its ability to scramble the elements from all over the original matrix using weighted sums. Thus, we propose a generalized structure for the RBT that allows arbitrarily sized RBTs and efficient distributed execution. Our generalized RBT takes the previously used formulation of RBT (herein called a *Parker RBT*) and truncates *each layer separately* to the desired size.

2 Previous Work

The RBT approach was first proposed in 1995 through a pair of tech reports by Parker [1995a,b] and a third tech report by Parker and Lê [1995]. They outline the approach and provide basic theoretical and experimental analysis. Their theoretical analysis shows that with probability 1, GENP will have only nonzero pivots in exact arithmetic. They then tested 11 matrices with sizes from $n = 32$ to $n = 512$. The RBT solver provided similar solutions to LINPACK for most of the

matrices; however, its solutions for ill-conditioned problems were less accurate than LINPACK. Due to a lack of optimizations, their implementation performed worse than LINPACK.

Then between 2008 and 2016, Baboulin et al. refined the RBT idea into a performant solver over numerous papers. The primary thrust of their work targeted many-core and heterogeneous systems for both nonsymmetric and symmetric-indefinite problems [Baboulin et al., 2008, 2013, 2015b, 2016; Becker et al., 2012; Tomov et al., 2010]. This line of work included several improvements, including

—truncating the transform's recursion depth to 2,
—designing efficient RBT kernels, and
—using iterative refinement.

Overall, their work shows excellent speedups over partial pivoting and reliably solved the test matrices, although the accuracy tests were limited to problems of size $n = 1{,}024$. Additionally, they explored the use of the RBT strategy for a distributed, symmetric-indefinite solver [Baboulin et al., 2014], for sparse factorization [Baboulin et al., 2015c], and for incomplete sparse factorizations [Baboulin et al., 2015a; Jamal et al., 2016].

Building on the work of Baboulin et al., Donfack et al. compared different replacements for partial pivoting (including the RBT), in a single-node, multicore setting [Donfack et al., 2015]. In that work, the speedup of the RBT solver compared to partial pivoting was much lower than in the GPU-based studies, likely because CPUs handle irregular work better than GPUs. But, more notably, they tested the accuracy of RBT with $n = 30{,}000$ and demonstrated that a depth-2 RBT fails to sufficiently transform all problems (specifically the `ris` and `orthog` matrices). We also previously extended this style of RBT to a distributed, heterogeneous LU-factorization [Lindquist et al., 2020]. Our implementation demonstrated large speedups over partial pivoting and accurate results for a few matrices of size $n = 100{,}000$. However, like Donfack et al., we found that the RBT is ineffective on large versions of the `orthog` matrix. (See Section 6.3 for analysis of using the RBT on `ris` and `orthog`.)

An interesting, recent work by Shen et al. combines RBTs with a modified version of the **adaptive cross approximation (ACA)** algorithm [Shen et al., 2022]. The ACA algorithm takes advantage of low-rank properties in the matrix's off-diagonal submatrices to factor a dense matrix in $O(n \log(n))$ time. However, intuition suggests these optimizations would work against each other since the RBT tries to equalize the rank between the on- and off-diagonal parts of the matrix while ACA relies on the off-diagonal submatrices being low-rank; this is likely the cause of their experimental loss of accuracy compared to the regular RBT solver. Unfortunately, their experiments are limited to matrices that can be factored without pivoting, and they did not test the performance of a non-RBT, nonpivoted factorization (with or without ACA), making it unclear whether the randomization actually benefited the accuracy.

Additionally, there has been recent work on the theoretical properties of RBTs [Peca-Medlin and Trogdon, 2023; Trogdon, 2019]. Those results show that preprocessing the identity matrix with RBTs results in a quadratic median growth for GENP (suggesting that RBTs will not significantly increase the growth for a given matrix). Furthermore, they experimentally show that, for $n = 256$, the majority of RBTs reduce the growth of GENP applied to Wilkinson's matrix [Wilkinson, 1965] from 10^{77} to less than 10^{6}. Unfortunately, they considered butterflies based on rotation matrices instead of the butterflies based on block-Hadamard matrices used by Baboulin et al. Peca-Medlin also developed a general-radix RBT, analogous to FFTs based on prime factorizations [Peca-Medlin, 2021]. This improves the flexibility of the RBT, but still requires the matrix's size and distribution to be highly composite with known divisors. Thus, our generalization provides a higher degree of flexibility, but the two approaches could also be combined.

Butterfly matrices are not the only transform that has been proposed for randomized preprocessing. First, Parker and Pierce proposed the "Randomizing Fast Fourier Transform," which combines an FFT with a random, diagonal scaling, as an alternative to the RBT [Parker and Pierce, 1995]. This is related to later sub-sampling transforms which randomly sample the columns of a randomizing FFT (although, later transforms are usually formulated as the transpose). Such transforms include the fast Johnson-Lindenstrauss transform [Ailon and Chazelle, 2006], the sub-sampled random Fourier transform [Woolfe et al., 2008], and the subsampled random Hadamard transform [Nguyen et al., 2009]. They proved that applying a randomizing FFT to either side of a matrix resulted in a strongly nonsingular matrix with probability 1. Pan et al. have also explored numerous transforms, including Gaussian and circulant matrices [Pan et al., 2015; Pan and Zhao, 2017]. They proved that a matrix preconditioned with one or more Gaussian matrices can be factored with a limited growth factor using GENP with high probability. Furthermore, experimental results with various preconditioning matrices show numerical errors of 10^{-10} to 10^{-15} with GENP, one step of iterative refinement, and matrices of size $n = 256$ to $n = 4{,}096$.

Finally, there exist other types of randomized preprocessing. A related idea to the random multipliers is to instead add a random matrix, which can be corrected with the Woodbury formula [Pan et al., 2013]. For low-rank problems, "sketching" techniques have recently become popular [Martinsson and Tropp, 2020] and are advancing rapidly [Murray et al., 2023, Sections 2, 7, and A]. These techniques multiply the system matrix by a random, rectangular matrix to reduce the problem to a smaller dimension, while retaining the large singular values.

Besides randomized preprocessing, several other strategies address the cost of pivoting in dense Gaussian elimination. Most notably, **Gaussian elimination with tournament pivoting (GETP)** selects a whole block of pivots at a time to reduce the number of synchronizations needed [Grigori et al., 2011]. Threshold pivoting also modifies the pivot selection process to allow smaller pivots that require less data movement [Lindquist et al., 2022]. Another strategy is dynamic pivoting, which avoids swapping rows in memory when pivoting and instead rearranges the matrix distribution [Geist and Romine, 1988]. Finally, **block elimination with additive modifications (BEAM)** replaces pivoting with additive perturbations to diagonal blocks during the factorization [Lindquist et al., 2023]; this reduces element growth in the Schur-complements but requires a correction with either iterative refinement or the Woodbury formula.

3 Generalized RBTs

RBTs are constructed by alternating random, diagonal matrices with orthogonal matrices. Our generalized formulation uses orthogonal butterfly matrices of the form

$$O_{\mu,\nu}^{\langle m\rangle} = \begin{bmatrix} \frac{1}{\sqrt{2}}I_\mu & 0 & \frac{1}{\sqrt{2}}I_\mu \\ 0 & I_\nu & 0 \\ \frac{1}{\sqrt{2}}I_\mu & 0 & \frac{-1}{\sqrt{2}}I_\mu \end{bmatrix}, \tag{2}$$

where $2\mu + \nu = m$ is the matrix dimension and I_μ is an identity matrix of size μ. The values of μ and ν will be chosen depending on the desired structure of the transform. These butterflies match Parker's butterflies when $\nu = 0$ or $m = 1$ [Parker, 1995a], so we call such matrices Parker butterflies. Interestingly, Figure 1 shows that these generalized butterflies still have a data dependency diagram that looks like a butterfly; the generalization merely adds the butterfly's body.

These individual butterfly matrices are composed with random diagonal matrices to create a depth-d RBT:

$$\mathcal{U}^{\langle n\rangle} = \operatorname{diag}\left(O_{\mu_{d,1},\nu_{d,1}}^{\langle \ell_{d,1}\rangle}, O_{\mu_{d,2},\nu_{d,2}}^{\langle \ell_{d,2}\rangle}, \dots\right) R_d \cdots O_{\mu_{1,1},\nu_{1,1}}^{\langle n\rangle} R_1, \tag{3}$$

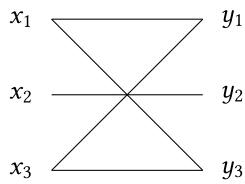

Fig. 1. Data dependencies for multiplying the vector $[x_1^T, x_2^T, x_3^T]^T$ by an orthogonal butterfly matrix to produce $[y_1^T, y_2^T, y_3^T]^T$.

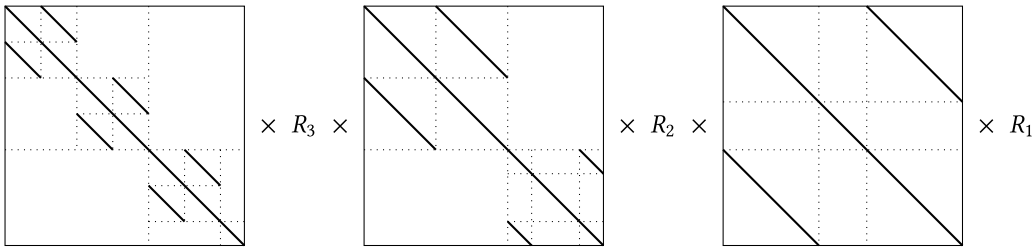

Fig. 2. The structure of one possible depth-3 RBT (specifically, a semi-Parker RBT).

where the sizes are chosen such that pairs of butterflies from one layer align with a single butterfly from the layer to the right. Previous formulations of the RBT (which we call Parker RBTs) also used Equation (3) except they limited the individual butterflies to being Parker butterflies. (Note that the butterflies in each pair can be switched using a fixed permutation matrix that depends on only the RBT structure. So, without loss of generality, we assume the first butterfly of each of these pairs is at least as large as the second.) In specific detail, if $\ell_{i,j}$ is the size of the jth butterfly (top to bottom) in the ith layer (right to left), the butterflies are chosen such that

$$\ell_{2i+1,2j} + \ell_{i+1,2j+1} = \ell_{i,j}, \qquad \ell_{2i+1,2j} = \mu_{ij} + \nu_{ij}, \qquad \text{and} \qquad \ell_{2i+1,2j+1} = \mu_{ij}. \tag{4}$$

Figure 2 demonstrates this structure. Note how the nice recursive structure of the RBT is retained while disconnecting the dimensions of the RBT layers from the global problem size. This removes the need to pad the matrix to a particular size. Furthermore, in the distributed case the sizes of the transforms can be aligned to the matrix distribution, which avoids complicated element-wise communication patterns [Baboulin et al., 2014; Lindquist et al., 2020].

Because Parker RBTs have a track record of providing accurate results, we ideally want to deviate from Parker RBTs by the minimum amount needed to obtain efficient execution. So, we define a *semi-Parker RBT* to be an RBT such that all the butterflies in Equation (3) are Parker butterflies except for the last block in each block-diagonal matrix. In other words, a semi-Parker RBT is like a Parker RBT, except each of the orthogonal layers is truncated to match the size of the system matrix. (However, a semi-Parker RBT is *not* equivalent to a Parker RBT that has been truncated after multiplying all the constituent matrices together.) We focus on semi-Parker RBTs, but non-semi-Parker RBTs would be useful for a matrix distributed with nonuniform tile sizes.

As a concrete example, Figure 3 shows an RBT selected to align with a matrix distributed in a 2D block-cyclic fashion. This is specifically a depth-2, semi-Parker RBT of size $n = 2{,}500$ with a tile size of 256. Because 768 is the smallest value that is both a multiple of 256 and greater than $2{,}500/2^d = 2{,}500/4$, it is used to choose the size of the butterflies. For the left term, the first butterfly is $O_{768,0}^{\langle 1{,}536 \rangle}$, a Parker butterfly. Then, the second butterfly has size $2{,}500 - 1{,}536 = 964$. Furthermore, to align with the tiles, we need $\mu_{2,2} = 964 - 768 = 196$. Thus, the second butterfly is $O_{196,572}^{\langle 964 \rangle}$.

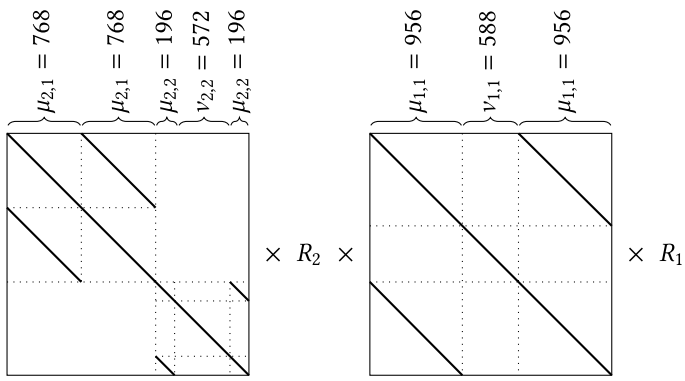

Fig. 3. The structure of a depth-2, semi-Parker RBT of size $n = 2{,}500$ chosen to align with a tile size of 256.

The right term easily follows. To align the lower rows with the second butterfly of the left term, we need $\mu_{1,1} = 964$, then $\nu_{1,1} = 2{,}500 - 2\mu_{1,1} = 588$. Hence, the right term is $O^{\langle 2{,}500 \rangle}_{956,588}$.

Like previous computational-focused works, we formulate our butterflies using random diagonal matrices and (truncated) block-Hadamard matrices. However, the theoretical analysis of Trogdon and Peca-Medlin uses butterflies based on block-rotation matrices. The idea of this generalization also applies to those matrices: simply use a rotation of 0° for the truncated portion. In other words, the individual butterfly matrices would take the form

$$\begin{bmatrix} C_\mu & 0 & S_\mu \\ 0 & I_\nu & 0 \\ S_\mu & 0 & -C_\mu \end{bmatrix}$$

with C, S being diagonal matrices such that $C_\mu^2 + S_\mu^2 = I_\mu$. Then, these butterfly matrices would be composed as in Equation (3), except with $R_i = I$. Such RBTs have the benefit of being orthogonal.

Generalized RBTs are implemented similarly to Parker RBTs, except the one-element rows of the transforms simply require scaling by the random factor. For a two-sided kernel, the loops must be split into four behaviors (each side can have either one or two elements). So, similar to previous formulations of the RBT, a depth-d transform can be applied to a vector in $2dn$ FLOP and to both sides of a matrix in $4dn^2$ FLOP when the normalization factors of Equation (3) are combined into the diagonal matrices. Furthermore, the storage of an RBT also remains at dn words.

Algorithm 1 presents the outline of an FFT-like application of a two-sided semi-Parker butterfly. This algorithm assumes U and V contain only the random coefficients, as per the packed storage used in previous works [Lindquist et al., 2020]. Line 3 controls the butterfly structure. For a nontiled implementation, setting $m \leftarrow 2^d \lceil 2^{-d} n \rceil$ gives the smallest m greater or equal to n that is a multiple of 2^d. For a tiled implementation (e.g., a distributed matrix) with a tile size of n_b, setting $m \leftarrow 2^d n_b \lceil 2^{-d} n_b^{-1} n \rceil$ ensures that the sizes of the constituent butterfly matrices are aligned to tile boundaries. Once m is chosen, the size of each butterfly and the corresponding loops easily follow. Line 4 loops over each layer one by one. Then, for each layer, lines 7–8 iterate over the Cartesian product of the butterflies matrices applied by that layer and apply each pair to the corresponding submatrix of A. This application of a single pair of butterflies and their corresponding diagonal matrices is described by the APPLY_BUTTERFLIES procedure. Note that, in the notation of Equation (2), μ for the left transform is m_{b2} and ν is $m_{b2} - m_{b1}$. Because of the differences between rows with 1 nonzero and those with 2, loop splitting is necessary and results in four loop bodies. Note that all of the loops, except the outermost one on line 4, can be completely parallelized. While Algorithm 1

Algorithm 1: Two-sided Semi-Parker RBT Application $\mathcal{U}^T A \mathcal{V}$ Using Packed Butterfly Storage

```
1:  procedure RBT(A ∈ ℝ^{n×n}, U ∈ ℝ^{n×d}, V ∈ ℝ^{n×d})
2:      d ← RBT depth
3:      m ← round up the dimension of A such that 2^{-d} m is nicely aligned
4:      for k from d to 1 do
5:          b_n ← 2^k                                          ▷ Number of butterflies
6:          h ← m/(2b_n)                                       ▷ Size of half of a butterfly
7:          for b_i from 1 to b_n do
8:              for b_j from 1 to b_n do
9:                  j_1 ← 2b_j h; j_2 ← j_1 + h; j_3 ← min(j_2 + h, n)   ▷ Col. indices for right butterflies
10:                 i_1 ← 2b_i h; i_2 ← i_1 + h; i_3 ← min(i_2 + h, n)   ▷ Row indices for left butterflies
11:                 APPLY_BUTTERFLIES(A[i_1:i_2, j_1:j_2], A[i_1:i_2, j_2:j_3], A[i_2:i_3, j_1:j_2], A[i_2:i_3, j_2:j_3],
                                      U[i_1:i_2, k], U[i_2:i_3, k], V[j_1:j_2, k], V[j_2:j_3, k])
12:             end for
13:         end for
14:     end for
15: end procedure

16: procedure APPLY_BUTTERFLIES(A_11, A_12, A_21, A_22, U_1, U_2, V_1, V_2)
17:     m_b1, n_b1 ← dim(A_11); m_b2, n_b2 ← dim(A_22)          ▷ Assume m_b1 ≥ m_b2 and n_b1 ≥ n_b2
18:     μ_m ← m_b2; ν_m ← m_b1 − m_b2; μ_n ← n_b2; ν_n ← n_b1 − n_b2
19:     for j from 1 to μ_n do
20:         for i from 1 to μ_m do
21:             a_11 ← A_11[i, j]; a_21 ← A_21[i, j]; a_12 ← A_12[i, j]; a_22 ← A_22[i, j]
22:             A_11[i, j] ← 2^{-1}(U_1[i]a_11V_1[j] + U_1[i]a_12V_2[j] + U_2[i]a_21V_1[j] + U_2[i]a_22V_2[j])
23:             A_12[i, j] ← 2^{-1}(U_1[i]a_11V_1[j] − U_1[i]a_12V_2[j] + U_2[i]a_21V_1[j] − U_2[i]a_22V_2[j])
24:             A_21[i, j] ← 2^{-1}(U_1[i]a_11V_1[j] + U_1[i]a_12V_2[j] − U_2[i]a_21V_1[j] − U_2[i]a_22V_2[j])
25:             A_22[i, j] ← 2^{-1}(U_1[i]a_11V_1[j] − U_1[i]a_12V_2[j] − U_2[i]a_21V_1[j] + U_2[i]a_22V_2[j])
26:         end for
27:         for i from μ_m + 1 to μ_m + ν_m do
28:             a_11 ← A_11[i, j]; a_12 ← A_12[i, j]
29:             A_11[i, j] ← 2^{-1/2}(U_1[i]a_11V_1[j] + U_1[i]a_12V_2[j])
30:             A_12[i, j] ← 2^{-1/2}(U_1[i]a_11V_1[j] − U_1[i]a_12V_2[j])
31:         end for
32:     end for
33:     for j from μ_n + 1 to μ_n + ν_n do
34:         for i from 1 to μ_m do
35:             a_11 ← A_11[i, j]; a_21 ← A_21[i, j]
36:             A_11[i, j] ← 2^{-1/2}(U_1[i]a_11V_1[j] + U_2[i]a_21V_1[j])
37:             A_21[i, j] ← 2^{-1/2}(U_1[i]a_11V_1[j] − U_2[i]a_21v_1[j])
38:         end for
39:         for i from μ_m + 1 to μ_m + ν_m do
40:             A_11[i, j] ← U_1[i]A_11[i, j]V_1[j]
41:         end for
42:     end for
43: end procedure
```

applies the butterflies one layer at a time, for small d it is possible to reorganize the algorithm to combine all d layers into a single pass over the matrix. This would reduce the data movement by a factor of d and avoid synchronization between layers, but would increase the computation by a factor of $2^d/d$ and require separate implementations for different d.

4 Stability of GENP after RBT-Preprocessing

Successfully solving a system of equations with GENP requires that the diagonal elements in the factors are nonzero. This occurs when each leading, principal submatrix is nonsingular; such a matrix is called *strongly nonsingular*. Parker proved that transforming a matrix on both sides with Parker RBTs of depth $\log_2(n)$ will result in a strongly nonsingular matrix in exact arithmetic with probability 1 [Parker, 1995a]. We redo his analysis and theorems for our generalized RBT. Unfortunately, the strategy used by Parker to prove that transformed matrices are strongly nonsingular does not apply to our formulation in its full generality, so we focus on semi-Parker RBTs instead. Our Lemmas 1 and 2 and Theorem 3 match Parker's Theorems 2–4 [Parker, 1995a], respectively. Technically, our results are weaker than Parker's since Parker proved that the transformed matrix is block nondegenerate (i.e., any permutation of the matrix is strongly nonsingular), but strong nonsingularity is sufficient for nonpivoted LU factorization.

For ease of comparison with Parker's theory, we generally follow his notation [Parker, 1995a]. The set of strictly increasing sequences of length k drawn from $\{1, 2, \ldots, m\}$ is denoted S_k^m. Furthermore, $i{:}j$ denotes the sequence of consecutive integers from i to j, inclusive. Addition and modulus are both applied element-wise. The length of a sequence, α, is denoted $|\alpha|$. The relation $\cong$ considers the sequences in sorted order and tests equality element-wise. (Note that this accounts for elements' multiplicity.) Similarly, the relation $\alpha \equiv \beta \pmod{m}$ applies the modulus to each element before comparing the sequences as per $\cong$. When a matrix is indexed with a pair of sequences, the result is the submatrix of the corresponding rows and columns.

The proofs use the Binet-Cauchy theorem extensively to distribute determinants across non-square matrix-multiplication, with the random coefficients preventing cancelation in the resulting summation. Thus, we reduce the question of whether the transformed matrix is strongly nonsingular to whether a large number of simple submatrices are nonsingular. First, Lemma 1 describes when submatrices of individual butterfly matrices are nonsingular. Then, Lemma 2 combines those individual butterfly matrices to show that the determinant of a submatrix of a semi-Parker RBT is a degree-1 polynomial of a subset of the random variables, with the subset depending on the submatrix indices. Furthermore, it says that, for a key set of submatrices, the polynomial is nonzero. Finally, Theorem 3 shows that if the transformed matrix is not strongly nonsingular, then the original matrix must be singular, which is the contrapositive of the desired result.

Lemma 1 (Cf. Parker's Theorem 2 [Parker, 1995a]). *For any $1 \le k \le n$, let $\alpha, \gamma \in S_k^n$ and let*

$$c(n, \alpha, \gamma) = det\left(O_{\mu,\nu}^{\langle n \rangle}[\alpha, \gamma]\right)$$

be a constant with $O_{\mu,\nu}^{\langle n \rangle}$ defined by Equation (2). Then, $c(n, \alpha, \gamma) = 0$ if and only if $\alpha \not\equiv \gamma \pmod{\mu + \nu}$. Otherwise, $2^{-|\alpha|/2} \le |c(n, \alpha, \gamma)| \le 1$.

Proof. First, note that $O_{\mu,\nu}^{\langle n \rangle}[i, j]$ is nonzero if and only if $i \equiv j \pmod{\mu + \nu}$ for any i, j.

To prove the backward implication, suppose that $\alpha \not\equiv \gamma \pmod{\mu + \nu}$. Then, there is a $j \in \alpha$ where either $j \notin \gamma \pmod{\mu + \nu}$ or the multiplicity of j in $\alpha \pmod{\mu + \nu}$ differs from that of j in $\gamma \pmod{\mu + \nu}$. In the former case, $O_{\mu,\nu}^{\langle n \rangle}[\alpha, \gamma]$ has a zero row, and thus, $c(n, \alpha, \gamma) = 0$. So, assume that $j \in \gamma \pmod{\mu + \nu}$ but with a different multiplicity than in α. If the multiplicity of j is 1 in $\alpha \pmod{\mu + \nu}$ but 2 in $\gamma \pmod{\mu + \nu}$, then the corresponding columns are both zero except for

a single nonzero in the jth row. Similarly, if the multiplicity of j is 2 in $\alpha \pmod{\mu+\nu}$ but 1 in $\gamma \pmod{\mu+\nu}$, then the corresponding rows are both zero except for a single nonzero in the jth column. Hence, we have linearly dependent columns or rows, and $c(n,\alpha,\gamma) = 0$.

Next, we prove other direction of the equivalence by its contrapositive. So, suppose that $\alpha \equiv \gamma \pmod{\mu+\nu}$. Thus, $O^{\langle n\rangle}_{\mu,\nu}[\alpha,\gamma]$ has either one or two nonzeros per row. Rows with one nonzero element simply scale the determinant by ± 1 or $\pm 2^{-1/2}$ and correspond to a column with no-other nonzero values. Rows with two nonzero elements exist in pairs with nonzeros present in the same columns and no other nonzeros in those columns. So, because that 2×2 submatrix is orthogonal, it scales the determinant by ± 1. Hence, $2^{-|\alpha|/2} \le |c(n,\alpha,\gamma)| \le 1$, which implies $c(n,\alpha,\gamma) \neq 0$. □

Lemma 2 (Cf. Parker's Theorem 3 [Parker, 1995a]). *Let $\mathcal{U}^{\langle n\rangle}$ be a semi-Parker RBT of size n with $d = \lceil \log_2(n)\rceil + 1$. Then for $1 \le k \le n$ and all sequences $\alpha, \beta \in S^n_k$, the determinant $det(\mathcal{U}^{\langle n\rangle}[\alpha,\beta])$ is a polynomial of degree at most one in each of the random variables of the RBT with coefficients based on α, β, and k. This polynomial is either zero or a function of a different (but possibly nondisjoint) set of random variables than $det(\mathcal{U}^{\langle n\rangle}[\gamma,\delta])$ for any other $\gamma,\delta \in S^n_k$. Furthermore, if $\beta = [1,2,\ldots,k]$, then $det(\mathcal{U}^{\langle n\rangle}[\alpha,\beta]) \neq 0$.*

Proof. We prove the theorem by induction on n. The base case $n = 1$ is trivial since $\mathcal{U}^{\langle 1\rangle}$ is a 1×1 matrix with a single random value satisfying all of the stated properties.

Let $\oplus$ denote the direct sum of two matrices, i.e.,

$$\mathcal{U}_0^{\langle \ell_0\rangle} \oplus \mathcal{U}_1^{\langle \ell_1\rangle} = \begin{bmatrix} \mathcal{U}_0^{\langle \ell_0\rangle} & 0 \\ 0 & \mathcal{U}_1^{\langle \ell_1\rangle} \end{bmatrix}.$$

Then, $\mathcal{U}^{\langle n\rangle} = \left(\mathcal{U}_0^{\langle \ell_0\rangle} \oplus \mathcal{U}_1^{\langle \ell_1\rangle}\right) O^{\langle n\rangle} R$ with $\ell_0 + \ell_1 = n$. Thus, by the Binet-Cauchy theorem [Marcus and Minc, 1992, p. 14],

$$\det(\mathcal{U}^{\langle n\rangle}[\alpha,\beta]) = \sum_{\gamma\in S^n_k} \det\left((\mathcal{U}_0^{\langle \ell_0\rangle} \oplus \mathcal{U}_1^{\langle \ell_1\rangle})\,[\alpha,\gamma]\right) \det\left(O^{\langle n\rangle}[\gamma,\beta]\right) \det\left(R[\beta,\beta]\right).$$

Furthermore, the determinant of $(\mathcal{U}_0^{\langle \ell_0\rangle} \oplus \mathcal{U}_1^{\langle \ell_1\rangle})[\alpha,\gamma]$ is zero unless there are subsequences α_0, α_1 of $1,\ldots,\ell_0$ and $(\ell_0+1,\ldots,n)$, respectively, whose concatenation is α and similar subsequences γ_0, γ_1 for γ such that $|\alpha_0| = |\gamma_0|$ and $|\alpha_1| = |\gamma_1|$. Then,

$$\det\left((\mathcal{U}_0^{\langle \ell_0\rangle} \oplus \mathcal{U}_1^{\langle \ell_1\rangle})\,[\alpha,\gamma]\right) = \det\left(\mathcal{U}_0^{\langle \ell_0\rangle}\,[\alpha_0,\gamma_0]\right) \det\left(\mathcal{U}_1^{\langle \ell_1\rangle}\,[\alpha_1-\ell_0,\gamma_1-\ell_0]\right).$$

First, by induction, these two smaller determinants are each a polynomial of degree at most one in their new (disjoint) sets of random variables. By Lemma 1, $\det(O^{\langle n\rangle}\,[\gamma,\beta])\det(R\,[\beta,\beta])$ is also a polynomial of degree at most one in its random variables, which are disjoint from those in $\mathcal{U}_0^{\langle \ell_0\rangle}$ and $\mathcal{U}_1^{\langle \ell_1\rangle}$. Hence, the determinant of $\left(\mathcal{U}_0^{\langle \ell_0\rangle} \oplus \mathcal{U}_1^{\langle \ell_1\rangle}\right)[\alpha,\gamma]$ is a polynomial of degree at most one in each of the random variables.

Second, by Equation (3), a depth $\lceil \log_2(n)\rceil + 1$ RBT can be factored as $\mathcal{U}^{\langle n\rangle} = IR^{\langle 1\rangle}PR^{\langle n\rangle}$, where $R^{\langle 1\rangle} = \mathrm{diag}(r_j^{\langle 1\rangle})$, $R^{\langle n\rangle} = \mathrm{diag}(r_j^{\langle n\rangle})$, and P is a product of the other matrices from Equation (3). So,

$$\det\left(\mathcal{U}^{\langle n\rangle}[\alpha,\beta]\right) = \left(\prod_{j\in\alpha} r_j^{\langle 1\rangle}\right) p(\alpha,\beta) \left(\prod_{j\in\beta} r_j^{\langle n\rangle}\right),$$

where $p(\alpha,\beta) = \det(P[\alpha,\beta])$. Note that $p(\alpha,\beta)$ is a polynomial not involving the random variables $r_j^{\langle 1\rangle}$ or $r_j^{\langle n\rangle}$. So, $\det(\mathcal{U}^{\langle n\rangle}[\alpha,\beta])$ is nonzero if and only if $p(\alpha,\beta)$ is nonzero. And if $\det(\mathcal{U}^{\langle n\rangle}[\alpha,\beta])$

is nonzero, then substituting any other pair of sequences for α, β will yield a function of a different (but possibly nondisjoint) set of random variables.

Third, let $\beta = [1, 2, \ldots, k]$. We will prove that $\det(\mathcal{U}^{\langle n \rangle}[\alpha, \beta]) \neq 0$ using induction. Continuing the factorization,

$$\mathcal{U}_0^{\langle \ell_0 \rangle} \oplus \mathcal{U}_1^{\langle \ell_1 \rangle} = R^{\langle 1 \rangle} Q R^{\langle \ell_0 + \ell_1 \rangle}$$

with $P = QR^{\langle \ell_0+\ell_1 \rangle} O^{\langle n \rangle}$. So, by Lemma 1 and the Binet-Cauchy theorem,

$$\begin{aligned}\det\left(\mathcal{U}^{\langle n \rangle}[\alpha, \beta]\right) &= \sum_{\gamma \in S_k^n} \det\left((\mathcal{U}_0^{\langle \ell_0 \rangle} \oplus \mathcal{U}_1^{\langle \ell_1 \rangle})\,[\alpha, \gamma]\right) c(n, \gamma, \beta) \left(\prod_{j \in \beta} r_j^{\langle n \rangle}\right) \\ &= \sum_{\gamma \in S_k^n} \left(\prod_{j \in \alpha} r_j^{\langle 1 \rangle}\right) \det(Q[\alpha, \gamma]) \left(\prod_{j \in \gamma} r_j^{\langle \ell_0+\ell_1 \rangle}\right) c(n, \gamma, \beta) \left(\prod_{j \in \beta} r_j^{\langle n \rangle}\right).\end{aligned}$$

The contribution to this sum by any given γ is thus either zero or a function of a different set of random variables than any other γ. Then, all that is needed to complete the proof is to find a value of γ for which both $\det\left((\mathcal{U}_0^{\langle \ell_0 \rangle} \oplus \mathcal{U}_1^{\langle \ell_1 \rangle})[\alpha, \gamma]\right)$ and $c(n, \gamma, \beta)$ are nonzero since no other γ produces a term with an identical set of random variables that could allow for cancelation.

If α_0, α_1 are subsequences of $1, \ldots, \ell_0$ and $\ell_0+1, \ldots, n$ whose sequence concatenation is α, then it is sufficient to show that there always exist *consecutive* subsequences γ_0, γ_1 of $1, \ldots, \ell_0$ and $\ell_0+1, \ldots, n$ such that γ is their concatenation, $c(n, \gamma, \beta) \neq 0$, $|\alpha_0| = |\gamma_0|$, and $|\alpha_1| = |\gamma_1|$. (Although, γ itself may be nonconsecutive.) One such pair of sequences is

$$\begin{aligned}\gamma_1 &= [1, 2, \ldots, |\alpha_1|] + \mu + \nu \\ \gamma_0 &= [|\alpha_1| + 1, \ldots, |\alpha_1| + |\alpha_0|] \bmod \mu + \nu.\end{aligned}$$

Recall that the partitioning of α requires $|\alpha_1| \leq \mu$. So, both γ_0 and γ_1 are trivially (wrap around) consecutive. Furthermore, $\gamma \equiv \beta \pmod{\mu + \nu}$, and Lemma 1 implies that $c(n, \gamma, \beta) \neq 0$. Because $\mathcal{U}^{\langle n \rangle}$ is a semi-Parker RBT, the butterflies in $\mathcal{U}_0^{\langle \ell_0 \rangle}[\alpha_0, \gamma_0]$ are all Parker butterflies and, thus, its size is a power of 2. Hence, by Parker's Theorem 3 and induction, respectively, the determinants of $\mathcal{U}_0^{\langle \ell_0 \rangle}[\alpha_0, \gamma_0]$ and $\mathcal{U}_1^{\langle \ell_1 \rangle}[\alpha_1, \gamma_1]$ are nonzero polynomials. □

Theorem 3 (Cf. Parker's Theorem 4 [Parker, 1995a]). *Let A be a nonsingular matrix, and let $\mathcal{U}^{\langle n \rangle}$ and $\mathcal{V}^{\langle n \rangle}$ be conformant, depth-$\lceil \log_2(n) \rceil + 1$, semi-Parker RBTs. Then, with probability 1, $\widetilde{A} = \mathcal{U}^{\langle n \rangle T} A \mathcal{V}^{\langle n \rangle}$ is strongly nonsingular (i.e., every leading principal submatrix is nonsingular).*

Proof. This proof closely follows that of Parker's Theorem 4 [Parker, 1995a], although, Parker formulates it as a proof by contradiction whereas we prove the contrapositive: If $\widetilde{A}$ is not strongly nonsingular, then A is singular.

Assume that $\widetilde{A}$ is not strongly nonsingular. Then, there is a $1 \leq k \leq n$ such that $\alpha_k = \{1, \ldots, k\}$ and $\det(\widetilde{A}[\alpha_k, \alpha_k] = 0)$. By the Binet-Cauchy theorem [Marcus and Minc, 1992, p. 14],

$$\det\left(\widetilde{A}[\alpha_k, \alpha_k]\right) = \sum_{\kappa \in S_k^n} \sum_{\lambda \in S_k^n} \det\left(\mathcal{U}^{\langle n \rangle}[\kappa, \alpha_k]\right) \det\left(A[\kappa, \lambda]\right) \det\left(\mathcal{V}^{\langle n \rangle}[\lambda, \alpha_k]\right). \tag{5}$$

By Lemma 2, $\det\left(\mathcal{U}^{\langle n \rangle}[\kappa, \alpha_k]\right)$ and $\det\left(\mathcal{V}^{\langle n \rangle}[\lambda, \alpha_k]\right)$ are nonzero polynomials of degree at most one of the randomization variables. Furthermore, each summand in Equation (5) is a function of a different (but possibly nondisjoint) set of randomization variables. Thus, $\det\left(\widetilde{A}[\alpha_k, \alpha_k]\right)$ is a polynomial of degree one in the randomization variables, and each $\det\left(\mathcal{U}^{\langle n \rangle}[\kappa, \alpha_k]\right) \det\left(\mathcal{V}^{\langle n \rangle}[\lambda, \alpha_k]\right)$

is a nonzero polynomial that depends on a different set of randomization variables. So, by the assumption on the lack of strong nonsingularity of $\widetilde{A}$ and Parker's Theorem 1 [Parker, 1995a], $\det(A[\kappa,\lambda]) = 0$ for all $\kappa,\lambda \in S_k^n$ with probability 1. However, the general Laplace expansion theorem for determinants [Marcus and Minc, 1992, p. 14] states that

$$\det(A) = \sum_{\lambda \in S_k^n} (-1)^{\sum_{i=1}^k \lambda_i} \det\left(A[\alpha_k,\lambda]\right) \det\left(A[(k{+}1{:}n),\lambda']\right),$$

where λ' is the complement of λ in $1:n$. Thus, $\det(A) = 0$. Therefore, if A is nonsingular, then $\widetilde{A}$ is strongly nonsingular with probability 1. □

To see the importance of the assumption that the RBTs are semi-Parker, note that

$$\mathcal{U}^{\langle 4\rangle} = R_1 \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} R_2 \begin{bmatrix} 1 & & 1 & \\ & 1 & & \\ 1 & & -1 & \\ & & & 1 \end{bmatrix} R_3 \begin{bmatrix} 1 & & & 1 \\ & 1 & & \\ & & 1 & \\ 1 & & & -1 \end{bmatrix} R_4$$

is a valid (non-semi-Parker) butterfly, but that $\det\left(\mathcal{U}^{\langle 4\rangle}[3:4,1:2]\right) = 0$. Thus, we cannot use Equation (5) to prove that if the transformed matrix is not strongly nonsingular, then all of the submatrices of A must have zero determinant, which is used to show that the original matrix is singular.

Unfortunately, Theorem 3 does not guarantee high accuracy in a finite-precision setting, even for semi-Parker RBTs. The primary source of instability is element growth that leads to cancelation errors. Using the Binet-Cauchy theorem, Parker suggested that the growth factor of the randomized matrix is, in some sense, the average growth over "all possible pivoting sequences (good and bad) with the original matrix" [Parker, 1995a, p. 14], which can be extended to our generalized formulation. Alternatively, using Schur's identity and the interlace theorem for singular values, we can show that

$$|\widehat{A}^{(k)}[i,j]| \le \frac{\sqrt{n}||\widehat{A}||_2}{\sigma_k(\widehat{A}[1:k,1:k])} \max_{ij} |\widehat{A}[i,j]|. \tag{6}$$

Thus, $\sigma_k(\widehat{A}[1:k,1:k])$ controls the growth.[1] For preprocessing with Gaussian transforms, Pan and Zhao have bounded $||\widehat{A}[1:k,1:k]||_2 = O(n^{-3/2})$ with high probability [Pan and Zhao, 2017, Theorem 4.3]; unfortunately, bounding it for RBT preprocessing is still an open problem.

For depths less than $\lceil \log_2(n) \rceil + 1$, not even theoretical strong nonsingularity is guaranteed. For example, consider

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},\ U = \begin{bmatrix} r_1 & 0 & r_3 & 0 \\ 0 & r_2 & 0 & r_4 \\ r_1 & 0 & -r_3 & 0 \\ 0 & r_2 & 0 & -r_4 \end{bmatrix},\ \text{and } V = \begin{bmatrix} r_5 & 0 & r_7 & 0 \\ 0 & r_6 & 0 & r_8 \\ r_5 & 0 & -r_7 & 0 \\ 0 & r_6 & 0 & -r_8 \end{bmatrix}. \tag{7}$$

Even though A is nonsingular (and even orthogonal), applying these depth-1 RBTs gives

$$U^T A V = \begin{bmatrix} 0 & 2r_1r_6 & 0 & 0 \\ 2r_2r_5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2r_3r_8 \\ 0 & 0 & 2r_4r_7 & 0 \end{bmatrix},$$

[1]This relation between the growth and the norm of the inverse of the leading principal submatrix provides valuable intuition on the growth factor of Gaussian Elimination in general; however, it has largely been ignored in the literature.

which is obviously not strongly nonsingular. However, experimental results have shown that a depth-2 transform can usually achieve a reasonable accuracy in practice [Baboulin et al., 2013; Becker et al., 2012].

5 Recovering Accuracy in the Solution Vector

As discussed in the previous section, this approach does not guarantee high accuracy, even probabilistically. Thus, a production-quality solver must be able to recover accuracy when the growth is too large. Iterative refinement is generally used for this purpose but relying upon it exclusively is often unsuccessful [Li and Demmel, 1998], especially the stationary formulation that can only recover minor errors. Such a refinement scheme can fail to converge if the inner solver is too inaccurate relative to the matrix's condition number [Carson and Higham, 2018]. Restarted GMRES can provide a more robust convergence at the cost of extra computation per iteration [Carson and Higham, 2017].

When refinement cannot recover enough accuracy, the problem must be re-solved with a more robust solver, such as GEPP, QR factorization, or even a mixture of both [Faverge et al., 2015]. This makes the solver much slower for problematic matrices; however, if the RBT solver is usually accurate and significantly faster in successful cases (as is shown in Section 6), the average time to solution will be lower than that of GEPP. Furthermore, in successful cases, this mechanism has no overhead since iterative refinement already needs a copy of the original matrix and checks the solution's accuracy. This fallback mechanism is an example of the "responsibly reckless" paradigm where a fast, reckless algorithm is combined with a responsible verification of the result [Dongarra et al., 2017].

Recent work on mixed-precision techniques advances the numerical analysis of the iterative refinement while fully utilizing the underlying hardware, using up to five floating-point precisions simultaneously to recover the lost digits in a method called GMRES-IR5 [Amestoy et al., 2024]. The use of these methods would further improve the accuracy of the solution in addition to our proposed generalized RBT. To make the numerical properties of our algorithm clear, we focus our evaluation on only the simplest form of accuracy-recovering techniques.

6 Experimental Results

To understand this generalization experimentally, we implemented an RBT-based solver in the **Software for Linear Algebra Targeting Exascale (SLATE)** library [Gates et al., 2019] and tested it on the Summit supercomputer. SLATE is a dense linear algebra library designed to replace ScaLAPACK in the increasingly heterogeneous landscape of high-performance computing. SLATE stores matrices in tiles which are distributed across the processes, usually in a 2D block-cyclic fashion. The initial implementation of the proposed generalized RBT is available in the 5 November 2023 release of SLATE, including the fallback strategy discussed in Section 5.

Our RBT implementation is based on Equation (3) and aligns the butterfly structure to these tiles in order to avoid the expensive tile management of other distributed implementations [Baboulin et al., 2014; Lindquist et al., 2020]. In other words, the top half of the first layer (i.e., the layer with a single butterfly matrix) is chosen to have a dimension equal to the first $2^{d-1}\lceil n_t 2^{-d}\rceil$ tiles where $n_t = \lceil n/n_b \rceil$ is the total number of rows of tiles in A. For simplicity, our implementation assumes that all tiles, except the last row and column, are the same size. For nonuniform tile sizes, it deviates from Equation (2) to keep the butterflies aligned to tiles; however, we did not consider the variable-tile-size case any further since we are unaware of any applications that effectively use nonuniform tiles in a distributed, dense factorization. As in previous works, the random variables are taken as $\exp(r/20)$ where r is uniformly selected from $[-1, 1]$ [Baboulin et al., 2013]. While our overall solver is GPU-enabled, our RBT implementation only uses the CPU. This choice is based on

the fact that the RBT kernel is inherently memory-bound and the MPI implementations available on Summit have to transfer data through the host.[2] Thus, any speedup gained by doing the computation on the GPU devices would be completely overshadowed by the overhead of transferring the data between the CPU memory and the attached GPU devices.

We test the generalized RBT solver against our GENP, GETP, and GEPP implementations. The first provides the best performance but is numerically unstable, while the other two provide better numerical stability but worse performance. Note that the performance of GEPP also provides bounds on the performance of threshold pivoting. Additionally, we compare it with Parker RBTs, based on our previous implementation [Lindquist et al., 2020]. Note that the fallback solver was always disabled to focus on the effects of the RBT alone. Furthermore, RBTs were always tested with a depth of 2 based on previous work showing that such a depth is usually sufficient.

All of our experiments (excluding a few numerical studies in Section 6.3) use the same basic experimental setup on the Summit supercomputer, which is outlined in Section 6.1. Our first pair of experiments, described in Section 6.2, considers the accuracy of our generalization when applied to various synthetic matrices and demonstrates that it retains the numerical benefit of Parker's RBTs. The first of these measures the accuracy of the solvers on various matrices of size $n = 150{,}000$, while the second tests a larger number of problem sizes on three select matrices. However, because the RBT was not universally successful on the tested matrices, Section 6.3 more carefully explores the numerics of three problematic matrices (`orthog`, `ris`, and `riemann`); low-rank submatrices in the original matrix are found to be the problem for the first two, but the third is only partially understood. After understanding the numerical accuracy of the method, we next turn to its performance in Section 6.4, which demonstrates the advantage of aligning the butterfly structure to the distribution. Finally, Section 6.5 compares the strong scaling of the RBT with that of pivoted factorizations.

6.1 Experimental Configuration

Our primary experiments were all performed on the Summit supercomputer at the Oak Ridge National Laboratory. Summit is based on IBM's POWER System AC922 node. Each node contains two 22-core, IBM POWER9 CPUs and six NVIDIA Volta V100 GPUs which are evenly divided between two sockets. Most of the computational capacity comes from the GPUs, each providing peak 7.45 TFLOP/s (80 SMs of 64 CUDA cores clocked at 1.455 GHz) and 16 GiB High Bandwidth Memory (HBM2) providing 900 GB/s peak main-memory bandwidth. Each CPU provides 540 GFLOP/s peak performance, 256 GiB DDR4 memory, and 170 GB/s peak main-memory bandwidth. Note that 1 core of each CPU is reserved for overheads associated with OS tasks and is not accessible by the user applications. NVLINK provides a peak bidirectional 50 GB/s transfer rate between the CPU and GPUs in a socket. A dual-rail EDR InfiniBand network connects the nodes with a bandwidth of 23 GB/s. All nodes use Red Hat Enterprise Linux version 8.2.

The code was compiled with GCC 9.1.0 and CUDA 11.0.3. It was linked against IBM's Spectrum MPI 10.4.0.3, ESSL 6.1.0-2, Netlib LAPACK 3.8.0, and Netlib ScaLAPACK 2.1.0. The code used for these experiments and the output files are available online at https://doi.org/10.6084/m9.figshare.24813789.

Each job is submitted to the scheduler with the flags `-nnodes 8 -alloc_flags smt1`; the second flag disables simultaneous multithreading. Processes were launched with ORNL-provided utility `jsrun -n 16 -a 1 -c 21 -g 3 -b packed:21 -d packed`, which allocates one process per socket and binds the threads to the corresponding CPUs and limits CUDA to the corresponding GPUs. The tester code from SLATE was used for all performance and accuracy results, with

[2]While the MPI implementations are capable of handling native GPU pointers, hardware limitations prevent GPU-Direct transfers between the GPU memories and the network without passing the data through the host on Summit.

Table 1. Comparison between the RBT Solver, GEPP, GETP, and GENP for the ∞-Norm Backward Error for Various Matrices of Size $n = 150{,}000$

Matrix	GEPP	GETP	GENP	RBT refined	RBT	Parker RBT refined	Parker RBT
`rand + nI`	2.0×10^{-14}	1.9×10^{-14}	1.8×10^{-14}	2.1×10^{-16}	2.0×10^{-14}	2.1×10^{-16}	1.8×10^{-14}
`rand`	2.5×10^{-14}	4.2×10^{-14}	2.4×10^{-10}	2.7×10^{-17}	2.1×10^{-10}	2.7×10^{-17}	7.0×10^{-11}
`rands`	5.6×10^{-14}	8.1×10^{-14}	2.7×10^{-10}	4.4×10^{-17}	1.3×10^{-9}	4.4×10^{-17}	1.6×10^{-9}
`randn`	5.3×10^{-14}	9.9×10^{-14}	6.8×10^{-10}	4.4×10^{-17}	3.4×10^{-10}	3.9×10^{-17}	3.5×10^{-10}
`randb`	4.0×10^{-14}	6.2×10^{-14}	NaN	2.2×10^{-17}	7.5×10^{-10}	2.5×10^{-17}	1.6×10^{-9}
`randr`	4.4×10^{-14}	7.5×10^{-14}	NaN	3.5×10^{-17}	1.6×10^{-9}	1.9×10^{-15}	4.9×10^{-8}
`chebspec`	3.4×10^{-16}	3.0×10^{-16}	2.0×10^{-9}	1.1×10^{-17}	9.8×10^{-14}	2.7×10^{-17}	2.8×10^{-13}
`circul`	1.4×10^{-17}	1.6×10^{-17}	9.5×10^{-14}	2.5×10^{-18}	1.0×10^{-17}	2.5×10^{-18}	9.0×10^{-18}
`fiedler`	2.1×10^{-18}	1.0×10^{-17}	NaN	2.3×10^{-18}	1.8×10^{-17}	2.0×10^{-18}	7.7×10^{-17}
`gfpp`	NaN	NaN	NaN	8.0×10^{-19}	3.7×10^{-17}	1.1×10^{-18}	1.8×10^{-17}
`orthog`	3.2×10^{-17}	3.2×10^{-17}	2.8×10^{-5}	1.0×10^{-3}	8.4×10^{-4}	8.4×10^{-4}	9.2×10^{-4}
`ris`	1.4×10^{-16}	9.8×10^{-17}	1.1×10^{-1}	1.5×10^{-1}	1.6×10^{-1}	1.4×10^{-1}	1.4×10^{-1}
`riemann`	4.1×10^{-14}	5.7×10^{-14}	2.8×10^{-12}	1.4×10^{-3}	1.5×10^{-3}	5.8×10^{-8}	1.2×10^{-3}
`zielkeNS`	3.5×10^{-19}	1.1×10^{-18}	NaN	8.9×10^{-19}	1.3×10^{-17}	1.1×10^{-18}	2.0×10^{-16}

a modification to print the accuracy as per Equation (8). We tuned SLATE's parameters using eight nodes and $n = 150{,}000$. All tests were configured with `--origin h --target d --type d --ref n --lookahead 2 --ib 64 --nrhs 1`. In other words:

- The matrix and vector data originate on the host.
- The majority of computation is done on the GPUs.
- Double precision is used for matrix and vector elements.
- The ScaLAPACK reference implementation is not run.
- Two lookahead tasks are used with depth 2 [Gates et al., 2019, Section 5].
- An inner blocking factor of 64 is used when factoring the panel or the diagonal tile.
- There was one right-hand side vector b and a corresponding single vector of unknowns x.

The matrix generator is configured with `--matrixB rand --seed 42 --seedB 64` plus `--matrix` option with the appropriate argument shown in the first column of Table 1;[3] this results in the same random right-hand side for each test with the same problem size and ensures that the system matrices are reproducible. GENP and the RBT-solvers were configured with `--nb 512 --fallback n`, i.e., a tile size of 512 and the fallback mechanism is disabled. Note that our tests always use the `gesv_rbt` routine for GENP results; overheads for the RBT and the iterative refinement are only present when the depth or iteration limit, respectively, is larger than 0. GEPP was configured with `--nb 896 --panel-threads 20`, i.e., a tile size of 896 and 20 threads to factor the panels. GETP was configured with `--nb 768 --panel-threads 2`, i.e., a tile size of 768 and 2 threads to do the tournament reduction. Finally, the flags `--check` and `--dim` were configured as appropriate for each test, as well as `--refine` and `--depth` for the GENP and RBT solvers. Because MPI and BLAS libraries often initialize their internal state and allocate temporary buffers on the first call, warm-up tests of size 10,000 were run first during each test where performance was measured. This avoided uniform overheads across or measurements without the influence of library initialization.

[3]The table names match the names in the SLATE tester, except for `rand + nI` which corresponds to `rand_dominant`.

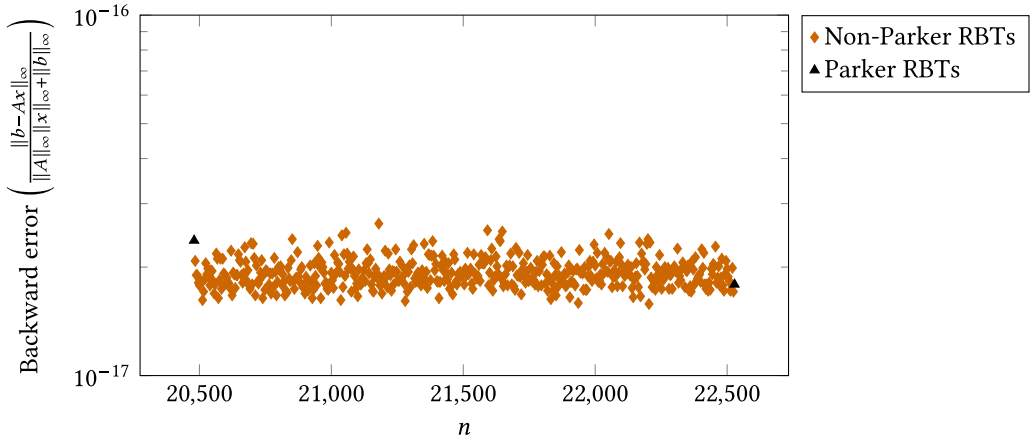

Fig. 4. Accuracy of the RBT-solver without iterative refinement for the various sizes of the `circul` matrix.

6.2 Accuracy

To demonstrate the accuracy of the approach, we tested the solvers with a variety of synthetic matrices and present the results in Table 1. The accuracy was measured with the ∞-norm backward error:

$$\frac{\|b - Ax\|_\infty}{\|A\|_\infty\|x\|_\infty + \|b\|_\infty}. \tag{8}$$

Problems were of size $150{,}000 = 292 \times 512 + 496$, which results in the generalized RBT method blocking the matrix differently than the original Parker RBT. Both formulations of the RBT solver were tested with and without two steps of iterative refinement. Additionally, using GENP with iterative refinement enabled it to achieve a double-precision solution in all cases except `orthog`, `ris`, and those with NaN values.[4] With iterative refinement, the butterfly solver provided similar or better accuracy than GEPP for 11 of the 14 matrices. It was even able to solve the `gfpp` matrix for which GEPP has catastrophic element growth that overflows double-precision. The three matrices with worse accuracy are analyzed in Section 6.3. Furthermore, any RBT provides significantly better accuracy than GENP for six matrices and provides significantly worse accuracy for only one matrix (`riemann`). Of the problems where two steps of iterative refinement did not reach double-precision accuracy, the refinement provided negligible benefit compared to the plain factorization in all cases except one (the `riemann` matrix with Parker RBT).

Additionally, the generalized formulation of the RBT gave similar accuracy to Parker's formulation. The one exception is that, for `riemann`, two steps of iterative refinement were able to improve the accuracy halfway to double-precision for Parker's formulation but not ours. (Continuing the iterative refinement allowed the Parker RBT-solver to reach double-precision accuracy after five iterations.) This difference is surprising, given the similarity in initial backward error; we suspect there is a subtle interaction between the RBT structure, its resulting errors, and the singular- or eigen-vectors of the matrix.

To further demonstrate the validity of our generalized RBT structure, Figures 4 to 6 show the accuracy of the RBT solver for three matrices of varying matrix sizes. These matrices are cases where the Parker RBT can improve their accuracy, even without iterative refinement. The first and last problems are multiples of 2,048 (i.e., 4 times the tile size), giving the strict recursive structure of Parker RBTs. Then, the generalized formula was tested with every intermediate multiple of 4. This

[4]The specific backward errors for GENP with iterative refinement can be found in our results artifact.

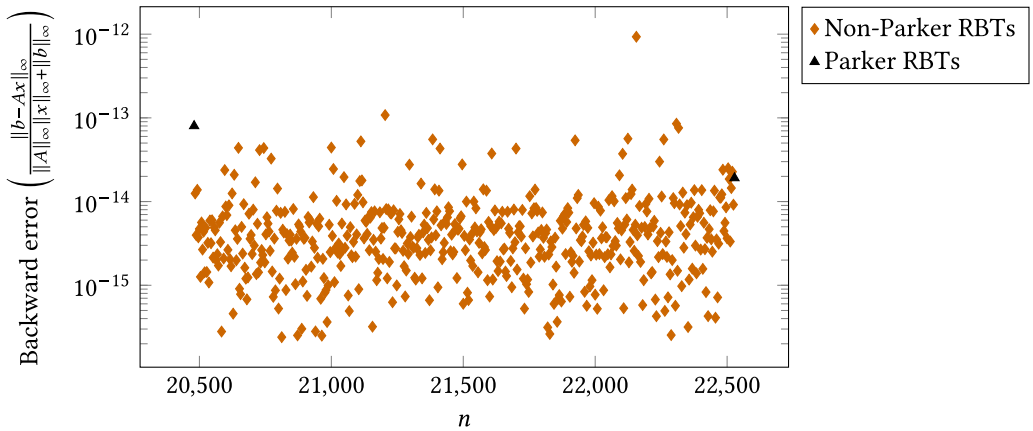

Fig. 5. Accuracy of the RBT-solver without iterative refinement for the various sizes of the chebspec matrix.

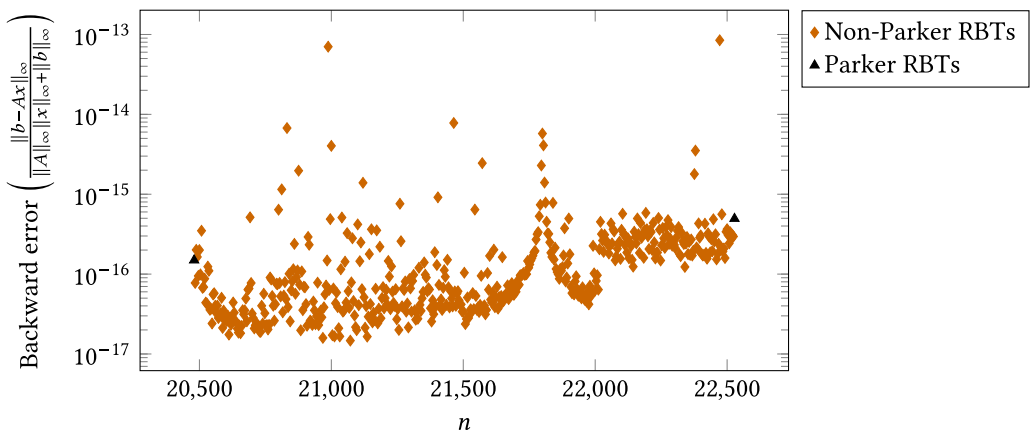

Fig. 6. Accuracy of the RBT-solver without iterative refinement for the various sizes of the `fiedler` matrix.

latter group represents all of the cases where a Parker RBT could be applied but our generalization has a different structure. Because we are interested specifically in the factorization error, no iterative refinement is applied in this test. So, if the generalized structure was problematically worse than Parker's formulation, the Parker cases would be noticeably better than the non-Parker cases (particularly the right side). However, most of the non-Parker cases are similar or better than the Parker cases. First, for `circul`, all the errors varied by less than a factor of 2, demonstrating little sensitivity to the RBT structure. Next, for `chebspec`, the errors varied significantly more without an obvious pattern. But, 94% of the non-Parker tests had a smaller error than both of the Parker tests, and only three of the non-Parker tests had a larger error than both of the Parker tests. Finally, `fiedler` is the most intriguing with visible trends in the data. Particularly, the error initially decreases from the left Parker RBT, spikes at around $n = 21{,}880$, then jumps up by half a digit around $n = 22{,}016$ (although there are numerous outliers, particularly on the left). Note that $n = 22{,}016$ corresponds to exactly 43 tile rows. Interestingly, the other two tile boundaries ($n = 20{,}992, 21{,}504$) do not have similar jumps in the accuracy. Fortunately, 63% of the non-Parker cases had a smaller error than both of the Parker cases, and only 6% were larger than both. So,

while our generalization seems to have some form of effect, it is not a significant overall reduction in stability.

6.3 Failures of the RBT Solver

Table 1 shows that the RBT solver matched the accuracy of GEPP in all but three matrices: `orthog`, `ris`, and `riemann`. Understanding these problems is important for understanding the weaknesses of the approach. Note that large element growth can only occur when one or more leading principal submatrices have small singular values (see Equation (6)). The `ris` matrix is the most straightforward; its entries are all close to zero except for a band along the anti-diagonal. Thus, the RBT needs to apply the anti-diagonal band to the first diagonal element, which requires a depth close to $\log_2(n)$. This behavior was distilled to Equation (7) as an example of how a depth less than $\lceil \log_2(n) \rceil$ is insufficient for strong nonsingularity.

The `orthog` matrix has a history of being challenging for GEPP alternatives [Becker et al., 2012; Donfack et al., 2015; Lindquist et al., 2020, 2022, 2023; Pan and Zhao, 2017]. This matrix is defined as

$$A[i, j] = \sqrt{\frac{2}{n+1}} \sin\left(\frac{i \times j \times \pi}{n+1}\right).$$

When $i \times j \ll n$, sine is approximately linear in its argument. Thus, the first few leading principal submatrices are close to rank-1 matrices, so plain GENP will have a very large growth factor. While the RBTs combine 2^{2d} submatrices to form the leading principal submatrix, it turns out that these submatrices are all numerically low rank. For the `orthog` matrix in Table 1, the 16 submatrices that combine to form the leading 512×512 principal submatrix together have only 266 singular values larger than 10^{-11} (with 6×10^{-2} being the largest element in the original matrix).[5] Furthermore, after applying the first orthogonal transform of the RBT to each side, the resulting four matrices together have only 147 singular values larger than 10^{-11} (if a Parker RBT is used instead, this decreases to 142 singular values). Thus, the leading principal block has many small singular values. This is fundamentally the same issue as in `ris` or Equation (7), except the rank deficiency is less obvious.

Finally, `riemann` is both the most concerning (since RBT-preprocessing reduces the accuracy, particularly the generalized formulation) and the most enigmatic. This matrix is related to the Riemann hypothesis and integer divisibility [Roesler, 1986]. Given the accuracy of GENP on this matrix and additional experimental results,[5] the leading principal submatrices of `riemann` appear full rank. However, if we apply a two-sided RBT where the random variables are set to 1, the resulting 256×256 leading principal submatrix has only 240 singular values larger than 10^{-8} (with 1.5×10^5 being the largest element in the original matrix). The random variables are supposed to prevent these types of cancelations, but this suggests that a larger range of random values may be useful for some matrices. Interestingly, although iterative refinement could converge for the Parker RBT but not the semi-Parker RBT, using a Parker RBT results in only 235 singular values larger than 10^{-8} for the leading 256×256 block. This suggests that there are likely additional factors limiting the accuracy of the RBT-solvers on this matrix, possibly relating to the singular- or eigen-vectors.

6.4 Performance

Figure 7 compares the performance of the generalized RBT solver with that of GEPP, GENP, and the old formulation of RBTs. The performance tests used `rand`, except for the diagonally dominant cases for GEPP and GETP, which used `rand + nI`. Each configuration was run four times, and the runtimes were summarized with the mean and the 99% CI; the performance was then computed

[5]See `submatrix-tests.jl` in the code artifact for more details.

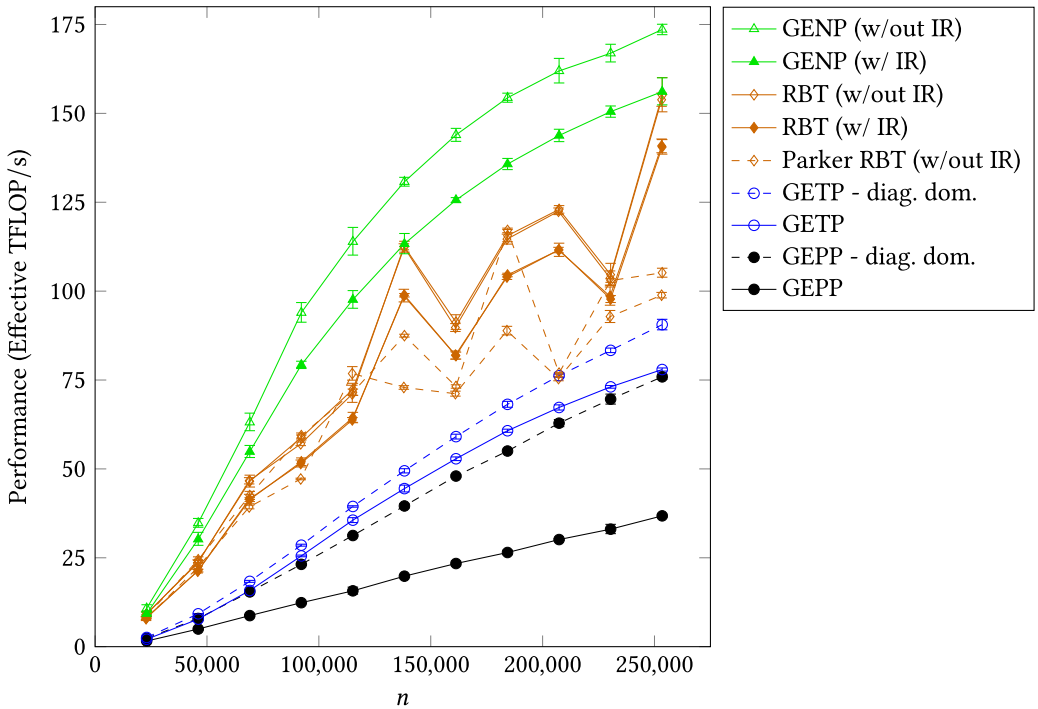

Fig. 7. Performance of the RBT solver versus GEPP and GENP. Each RBT configuration was tested with two sets of sizes that differ by half of a tile width (i.e., by 256). The performance is computed based on $\frac{2}{3}n^3$FLOP, to normalize the performance results. Error bars indicate the 99% CIs based on the runtime.

as a $\frac{2}{3}n^3$ FLOP count divided by either the mean, the lower CI bound, or the upper CI bound in seconds. All of the solvers were tested with matrices of dimension $23{,}040 \times i$ for $1 \leq i \leq 11$. Because $23{,}040 = 45 \times 512$, the Parker RBT will be aligned to whole tiles only every fourth matrix size. Additionally, the RBT solvers were tested with matrices smaller by half of a tile, which always corresponds to cases in which the Parker RBT solver is not aligned to whole tiles. We did not include a set of matrices where the Parker RBT is always aligned to tiles since, in that case, the generalized and Parker RBTs are the same. GENP and the RBT solvers were tested both with and without iterative refinement. For clarity's sake, the results for the Parker RBT with iterative refinement are omitted from Figure 7 but can be found in our results' artifact.

GENP and the RBT solvers provided noticeably higher performance than GEPP and GETP, although the RBT solver was about 30% slower than GENP. For the largest two problem sizes, the RBT solver with iterative refinement was 40% and 85% faster than the best case of GEPP. These speedups double when extensive pivoting is required. Thus, even if the RBT solver must fall back to GEPP 20% of the time, it will be on average more than 25% faster.

When comparing the two RBT formulations, the generalized RBT outperforms the Parker RBT by up to 62% and only underperforms on one problem size. However, for problem sizes like $n = 184{,}320$ where the Parker RBT aligns to the tiles, the two formulations provide identical performance. Because the generalized RBT always aligns itself to whole tiles, it performs identically between the two sets of problem sizes. On the other hand, the Parker RBT saw up to a 32% increase in runtime when decreasing the matrix size by 256 rows. Despite better tile alignment, the performance of the generalized RBT does not change smoothly with the problem size; the variation depends on

whether communication is needed for one or both of the layers in the transform. Because this test uses a 4×4 process grid, no communication is needed if and only if $\lceil n_t 2^{-d} \rceil$ is divisible by 4. In Figure 7, this condition occurs for $n = 23{,}040, n = 138{,}240, n = 253{,}440$, corresponding to the large spikes in the performance. Similarly, only one layer of the RBT communicates for $n = 69{,}120, n = 184{,}320, n = 207{,}360$, corresponding to the middle band of performance results. This communication overhead results in a 48% increase in the GFLOP/s rate, compared to only a 4% increase in that metric for GENP; this corresponds to a 10% *decrease* in runtime when increasing the problem size by 10% (and the FLOP count by 33%). This suggests that it is preferable to use a butterfly structure where the top half of the first layer (i.e., the layer with a single butterfly matrix) has dimension

$$\mathrm{lcm}(p,q)2^{d-1}\lceil n_t 2^{-d}\mathrm{lcm}(p,q)^{-1}\rceil \tag{9}$$

for a $p \times q$ process grid and $\mathrm{lcm}(p,q)$ being the least common multiple of p and q.

Both the RBTs and iterative refinement added a noticeable overhead, 12% and 11%, respectively, when $n = 253{,}440$; the former increases to 60% for the communication intensive $n = 230{,}400$. When compared to Parker RBTs, the new generalized transforms decreased the runtime of the *overall solver* by up to 38%; it did introduce a slowdown in two cases ($n = 46{,}080$ and $n = 92{,}160$) but these slowdowns were less than 10%.

As discussed in Section 5, when iterative refinement cannot recover sufficient accuracy the problem must be re-solved with a more robust solver. If we assume the RBT-solver fails for one in three matrices (which is worse than Table 1), our performance results indicate that the RBT-solver will outperform GEPP on *average*.[6] Furthermore, if the matrices are factored by GEPP in a time close to that of `rand`, the RBT solver will have an average speedup between 54% and 75%. Tournament pivoting is more competitive; when both layers of the RBT need to communicate between nodes, the RBT solver is 3% to 12% slower. But, if RBT does not require inter-node communication, the RBT solver is 9% to 20% faster.

6.5 Strong Scaling

In some applications, the linear systems are small relative to the number of nodes needed for the rest of the application [Ghysels and Synk, 2022]. Thus, strong scaling is important for achieving good performance in those applications. Toward that end, we tested the RBT approach for problems of size $n = 100{,}000$ with varying numbers of processes, shown in Figure 8. Grids were either $p \times p$ or $p \times 2p$, depending on the number of processes. (Recall that we used two processes per node due to the low inter-socket bandwidth.) Because SLATE's GETP allocates extra device memory for cuSOLVER to factor the local panels, it runs out of memory for the single node run.

Overall, the nonpivoted factorizations scale better than the pivoted ones, with GEPP showing a slowdown when going from one to two nodes. Note that GEPP used 1×2 and 2×2 process grids for one and two nodes, respectively, so the pivot search of the latter case introduces a distributed reduction for each column. A major factor in the worse scaling is that pivoting introduces data dependencies that reduce the available parallelism. For example, particularly, the block column (i.e., the panel) must be updated before the diagonal block can be applied to the block row; on the other hand, GENP can do these updates simultaneously. Interestingly, intuition says that increasing the number of processes should increase the number of inter-node row swaps, but Figure 8 shows worse scaling for the diagonally dominant matrices than for the pivoted matrices, suggesting that overhead for swapping rows parallelizes well.

[6]Details are provided in our data artifact.

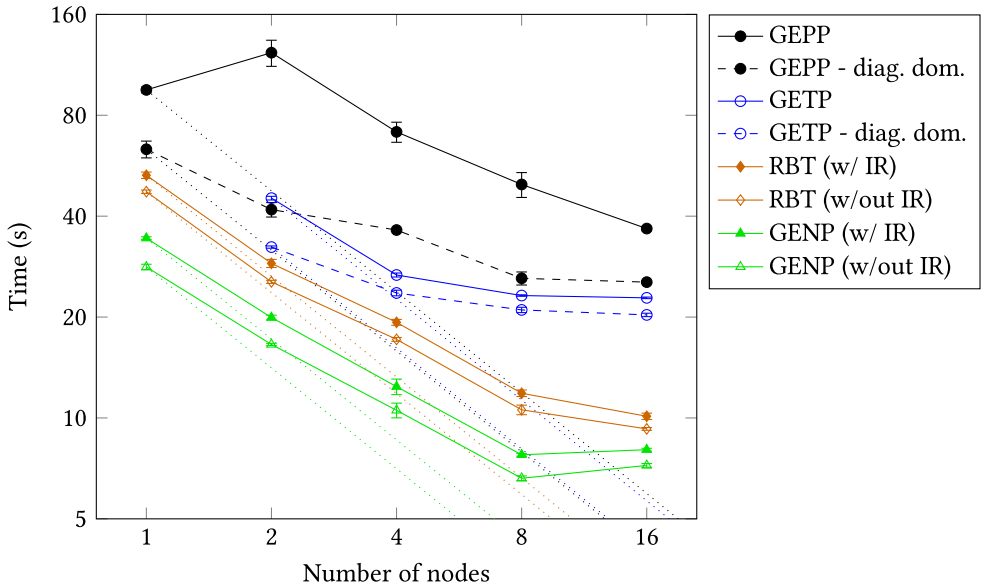

Fig. 8. Strong scaling of various solvers for a problem of size $n = 100{,}000$. The dotted lines show ideal scaling. Error bars show 99% CIs.

Interestingly, the ratio of the performance results of the RBT solver and GENP remains approximately constant from 1 to 8 nodes, but going to 16 nodes slightly improves the performance of the former while reducing that of the latter. This suggests that applying the RBTs scales well enough to partially offset the slowdown in the factorization.

7 Conclusions

We have developed a generalized version of the RBT that can be adapted to a matrix size and distribution, instead of needing to adapt the matrix to the solver. This formulation achieves the same theoretical result as Parker's original formulation and similar experimental accuracy at a lower cost. The key observation was that the RBT is simply a tool to randomize the matrix and that it should be modified to fit the application instead of modifying the application to fit the RBT. This contrasts with the previous work (including our own [Lindquist et al., 2020]) which took the restriction on matrix size as a given and either ignored it without comment or suggested users enlarge their matrix to fit the RBT.

An important observation from Section 6.3 is that the matrices for which the RBT-solver struggles to solve accurately are problems that have large regions with low rank, either originally (as in `ris` and `orthog`) or after applying the RBT (as in `riemann`). This suggests that future research to improve the robustness of RBT preconditioning should consider adding an unstructured reordering to the matrices. It has previously been suggested to combine RBT preprocessing with a technique like BEAM that fixes deficient diagonal elements with local information [Lindquist et al., 2023]. However, the observations here would suggest that local corrections will not provide a significant benefit. (In the case of BEAM, we expect that the additive perturbation would still provide a benefit on top of the RBT.)

There are several areas where this work can be extended. First, as noted in Section 6, the advantages of our generalized formulation still assume a uniform tile size. While we are unaware of applications that use nonuniform tile sizes with pivoted LU factorization, a further generalization

could be explored where the columns and rows of the individual butterfly matrices from Equation (2) are permuted. Our implementation supports such a generalization, but we have not investigated its use. Second, our current RBT kernel only runs on CPU. However, some recent systems, such as the Frontier supercomputer, have the network cards attached to the GPUs; for such systems, doing the RBT on the GPUs would reduce data movement. Furthermore, if the RBT size was chosen to eliminate all internode communication, the overhead to transfer remote data to GPU is removed, making a GPU-based RBT more effective. Thus, it would be valuable to design and test a GPU version of our generalized RBT.

In addition to their use in solving systems of linear equations, butterfly transforms are used in neural networks as a replacement for pointwise convolutions [Alizadeh Vahid et al., 2020], dense layers [Dao et al., 2019], and attention mechanisms [Fan et al., 2022]. However, the butterfly structure constrains the architecture of networks using those layers. Furthermore, the butterfly structure is used only to provide a cheap all-to-all transform, which our generalized formulation still provides. So, our generalized butterfly structure would allow more flexibility in the network architectures.

Finally, many other randomized linear algebra algorithms use random rectangular "sketching" matrices, such as those for least squares problems and low-rank approximations [Martinsson and Tropp, 2020]. One such family of random matrices is **subsampled random Fourier transforms (SRFTs)**, which randomly sample the rows of an FFT or fast cosine transform matrix after modifying the columns with random signs and permutations. However, they can be more expensive than other sparse random matrices when the sketched dimension is large [Murray et al., 2023, Section 2.5]. However, the experimental success of a depth-2 RBT for most matrices suggests that it sketches the matrix such that each $k \times k$ leading principal submatrix has rank k. Furthermore, there has been basic experimental success with one-sided RBTs [Baboulin et al., 2015c], which is even closer to the action of a sketching matrix. Thus, it is worth investigating an SRFT or RBT sketching matrix that uses a depth smaller than $\log(n)$ and that takes the leading rows instead of randomly subsampling. The latter simplification could also reduce the computational cost of sketching an $m \times n$ matrix to $O(mn)$ because half of the rows could be discarded after each step. The theoretical analysis of such a transform may not be as strong as traditional SRFTs, but the performance benefits may outweigh that.

References

Nir Ailon and Bernard Chazelle. 2006. Approximate Nearest Neighbors and the Fast Johnson-Lindenstrauss Transform. In *the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*. ACM, New York, NY, 557–563. DOI: https://doi.org/10.1145/1132516.1132597

Keivan Alizadeh Vahid, Anish Prabhu, Ali Farhadi, and Mohammad Rastegari. 2020. Butterfly Transform: An Efficient FFT Based Neural Architecture Design. In *2020 IEEE/CVF* Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Seattle, WA, USA, 12021–12030. DOI: https://doi.org/10.1109/CVPR42600.2020.01204

Patrick Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Theo Mary, and Bastien Vieublé. 2024. Five-Precision GMRES-Based Iterative Refinement. *SIAM Journal on Matrix Analysis and Applications* 45, 1 (Mar. 2024), 529–552. DOI: https://doi.org/10.1137/23M1549079

Marc Baboulin, Dulceneia Becker, George Bosilca, Anthony Danalis, and Jack Dongarra. 2014. An Efficient Distributed Randomized Algorithm for Solving Large Dense Symmetric Indefinite Linear Systems. *Parallel Computing* 40, 7 (Jul. 2014), 213–223. DOI: https://doi.org/10.1016/j.parco.2013.12.003

Marc Baboulin, Jack Dongarra, Julien Herrmann, and Stanimire Tomov. 2013. Accelerating Linear System Solutions Using Randomization Techniques. *ACM Transactions on Mathematical Software* 39, 2 (Feb. 2013), 1–13. DOI: https://doi.org/10.1145/2427023.2427025

Marc Baboulin, Jack Dongarra, Andrien Rémy, Stanimire Tomov, and Ichitaro Yamazaki. 2016. Dense Symmetric Indefinite Factorization on GPU Accelerated Architectures. In *Parallel Processing and Applied Mathematics (PPAM '15)*, Lecture Notes in Computer Science, Vol. 9573, Springer, Krakow, Poland, 86–95. DOI: https://doi.org/10.1007/978-3-319-32149-3_9

Marc Baboulin, Aygul Jamal, and Masha Sosonkina. 2015a. Using Random Butterfly Transformations in Parallel Schur Complement-Based Preconditioning. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, L'od'z, Poland, 649–654. DOI: https://doi.org/10.15439/2015F177

Marc Baboulin, Amal Khabou, and Adrien Rémy. 2015b. A Randomized LU-Based Solver Using GPU and Intel Xeon Phi Accelerators. In *Parallel Processing Workshops (Euro-Par '15)*. Sascha Hunold, Alexandru Costan, Domingo Giménez, Alexandru Iosup, Laura Ricci, María Engracia Gómez Requena, Vittorio Scarano, Ana Lucia Varbanescu, Stephen L. Scott, Stefan Lankes, Josef Weidendorfer, and Michael Alexander (Eds.), Lecture Notes in Computer Science, Springer, Vienna, Austria, 175–184. DOI: https://doi.org/10.1007/978-3-319-27308-2_15

Marc Baboulin, Xiaoye S. Li, and François-Henry Rouet. 2015c. Using Random Butterfly Transformations to Avoid Pivoting in Sparse Direct Methods. In *High Performance Computing for Computational Science – VECPAR '14*. Michel Daydé, Osni Marques, and Kengo Nakajima (Eds.), Lecture Notes in Computer Science, Springer, Eugene, OR, 135–144. DOI: https://doi.org/10.1007/978-3-319-17353-5_12

Marc Baboulin, Stanimire Tomov, and Jack Dongarra. 2008. Some Issues in Dense Linear Algebra for Multicore and Special Purpose Architectures. In *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA '08)*, Vol. 6126–6127, 1–12. Springer-Verlag, Trondheim, Norway.

Dulceneia Becker, Marc Baboulin, and Jack Dongarra. 2012. Reducing the Amount of Pivoting in Symmetric Indefinite Systems. In *Parallel Processing and Applied Mathematics*. Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy WaŚniewski (Eds.), Springer, Berlin, 133–142. DOI: https://doi.org/10.1007/978-3-642-31464-3_14

Erin Carson and Nicholas J. Higham. 2017. A New Analysis of Iterative Refinement and Its Application to Accurate Solution of Ill-Conditioned Sparse Linear Systems. *SIAM Journal on Scientific Computing* 39, 6 (Jan. 2017), A2834–A2856. DOI: https://doi.org/10.1137/17M1122918

Erin Carson and Nicholas J. Higham. 2018. Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. *SIAM Journal on Scientific Computing* 40, 2 (Jan. 2018), A817–A847. DOI: https://doi.org/10.1137/17M1140819

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Re. 2019. Learning Fast Algorithms for Linear Transforms Using Butterfly Factorizations. In *the 36th International Conference on Machine Learning*. PMLR, Long Beach, CA, 1517–1527.

Simplice Donfack, Jack Dongarra, Mathieu Faverge, Mark Gates, Jakub Kurzak, Piotr Luszczek, and Ichitaro Yamazaki. 2015. A Survey of Recent Developments in Parallel Implementations of Gaussian Elimination. *Concurrency and Computation: Practice and Experience* 27, 5 (Jun. 2015), 1292–1309. DOI: https://doi.org/10.1002/cpe.3306

Jack Dongarra, Stanimire Tomov, Piotr Luszczek, Jakub Kurzak, Mark Gates, Ichitaro Yamazaki, Hartwig Anzt, Azzam Haidar, and Ahmad Abdelfattah. 2017. With Extreme Computing, the Rules Have Changed. *Computing in Science & Engineering* 19, 3 (May 2017), 52–62. DOI: https://doi.org/10.1109/MCSE.2017.48

Hongxiang Fan, Thomas Chau, Stylianos I. Venieris, Royson Lee, Alexandros Kouris, Wayne Luk, Nicholas D. Lane, and Mohamed S. Abdelfattah. 2022. Adaptable Butterfly Accelerator for Attention-Based NNs via Hardware and Algorithm Co-Design. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Chicago, IL, 599–615. DOI: https://doi.org/10.1109/MICRO56248.2022.00050

Mathieu Faverge, Julien Herrmann, Julien Langou, Bradley Lowery, Yves Robert, and Jack Dongarra. 2015. Mixing LU and QR Factorization Algorithms to Design High-Performance Dense Linear Algebra Solvers. *Journal of Parallel and Distributed Computing* 85 (Nov. 2015), 32–46. DOI: https://doi.org/10.1016/j.jpdc.2015.06.007

Mark Gates, Jakub Kurzak, Ali Charara, Asim YarKhan, and Jack Dongarra. 2019. SLATE: Design of a Modern Distributed and Accelerated Linear Algebra Library. In *the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19)*. ACM, Denver, CO, 1–18. DOI: https://doi.org/10.1145/3295500.3356223

George A. Geist and Charles H. Romine. 1988. LU Factorization Algorithms on Distributed-Memory Multiprocessor Architectures. *SIAM Journal on Scientific and Statistical Computing* 9, 4 (Jul. 1988), 639–649. DOI: https://doi.org/10.1137/0909042

Pieter Ghysels and Ryan Synk. 2022. High Performance Sparse Multifrontal Solvers on Modern GPUs. *Parallel Computing* 110 (May 2022), 102897. DOI: https://doi.org/10.1016/j.parco.2022.102897

Laura Grigori, James W. Demmel, and Hua Xiang. 2011. CALU: A Communication Optimal LU Factorization Algorithm. *SIAM Journal on Matrix Analysis and Applications* 32, 4 (Oct. 2011), 1317–1350. DOI: https://doi.org/10.1137/100788926

Aygul Jamal, Marc Baboulin, Amal Khabou, and Masha Sosonkina. 2016. A Hybrid CPU/GPU Approach for the Parallel Algebraic Recursive Multilevel Solver pARMS. In *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, Timisoara, Romania, 411–416. DOI: https://doi.org/10.1109/SYNASC.2016.069

Xiaoye S. Li and J. W. Demmel. 1998. Making Sparse Gaussian Elimination Scalable by Static Pivoting. In *the 1998 ACM/IEEE Conference on Supercomputing (SC '98)*. IEEE Computer Society, San Jose, CA, 34–34. DOI: https://doi.org/10.1109/SC.1998.10030

Neil Lindquist, Mark Gates, Piotr Luszczek, and Jack Dongarra. 2022. Threshold Pivoting for Dense LU Factorization. In *2022 IEEE/ACM Workshop on Latest Advances in Scalable Algorithms for Large-Scale Heterogeneous Systems (ScalAH)*. IEEE Computer Society, Dallas, Texas, 34–42. DOI: https://doi.org/10.1109/ScalAH56622.2022.00010

Neil Lindquist, Piotr Luszczek, and Jack Dongarra. 2020. Replacing Pivoting in Distributed Gaussian Elimination with Randomized Techniques. In *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*. IEEE Press, Atlanta, GA, 35–43. DOI: https://doi.org/10.1109/ScalA51936.2020.00010

Neil Lindquist, Piotr Luszczek, and Jack Dongarra. 2023. Using Additive Modifications in LU Factorization Instead of Pivoting. In *the 37th ACM International Conference on Supercomputing*. ACM, Orlando, FL, 14–24. DOI: https://doi.org/10.1145/3577193.3593731

Marvin Marcus and Henryk Minc. 1992. *A Survey of Matrix Theory and Matrix Inequalities*, Vol. 14. Dover Publications, Inc, New York, NY.

Per-Gunnar Martinsson and Joel A. Tropp. 2020. Randomized Numerical Linear Algebra: Foundations and Algorithms. *Acta Numerica* 29 (May 2020), 403–572. DOI: https://doi.org/10.1017/S0962492920000021

Riley Murray, James Demmel, Michael W. Mahoney, N. Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michał Dereziński, Miles E. Lopes, Tianyu Liang, Hengrui Luo, and Jack Dongarra. 2023. Randomized Numerical Linear Algebra: A Perspective on the Field with an Eye to Software. Retrieved from https://doi.org/10.48550/arXiv.2302.11474

Nam H. Nguyen, Thong T. Do, and Trac D. Tran. 2009. A Fast and Efficient Algorithm for Low-Rank Approximation of a Matrix. In *the 41st Annual ACM Symposium on Theory of Computing (STOC '09)*. ACM, New York, NY, 215–224. DOI: https://doi.org/10.1145/1536414.1536446

Victor Y. Pan, Guoliang Qian, and Xiaodong Yan. 2015. Random Multipliers Numerically Stabilize Gaussian and Block Gaussian Elimination: Proofs and an Extension to Low-Rank Approximation. *Linear Algebra and its Applications* 481 (Sept. 2015), 202–234. DOI: https://doi.org/10.1016/j.laa.2015.04.021

Victor Y. Pan, Guoliang Qian, and Ai-Long Zheng. 2013. Randomized Preprocessing versus Pivoting. *Linear Algebra and its Applications* 438, 4 (Feb. 2013), 1883–1899. DOI: https://doi.org/10.1016/j.laa.2011.02.052

Victor Y. Pan and Liang Zhao. 2017. Numerically Safe Gaussian Elimination with No Pivoting. *Linear Algebra and its Applications* 527 (Aug. 2017), 349–383. DOI: https://doi.org/10.1016/j.laa.2017.04.007

D. Stott Parker. 1995a. *Random Butterfly Transformations with Applications in Computational Linear Algebra*. Technical Report CSD-950023. Computer Science Department, UCLA, Los Angeles, CA, 20 pages.

D. Stott Parker. 1995b. *A Randomizing Butterfly Transformation Useful in Block Matrix Computations*. Technical Report CSD-950024. Computer Science Department, UCLA, Los Angeles, CA, 20 pages.

D. Stott Parker and Dinh Lê. 1995. *How to Eliminate Pivoting from Gaussian Elimination — By Randomizing Instead*. Technical Report CSD-950022. Computer Science Department, UCLA, Los Angeles, CA, 14 pages.

D. Stott Parker and Brad Pierce. 1995. *The Randomizing FFT: An Alternative to Pivoting in Gaussian Elimination*. Technical Report CSD-950037. University of California, Los Angeles, CA.

John Peca-Medlin. 2021. *Numerical, Spectral, and Group Properties of Random Butterfly Matrices*. Ph.D. Dissertation. UC Irvine.

John Peca-Medlin and Thomas Trogdon. 2023. Growth Factors of Random Butterfly Matrices and the Stability of Avoiding Pivoting. *SIAM Journal on Matrix Analysis and Applications* 44, 3 (Sept. 2023), 945–970. DOI: https://doi.org/10.1137/22M148762X

Friedrich Roesler. 1986. Riemann's Hypothesis as an Eigenvalue Problem. *Linear Algebra and its Applications* 81 (Sept. 1986), 153–198. DOI: https://doi.org/10.1016/0024-3795(86)90255-7

Zhongyu Shen, Jilin Zhang, and Tomohiro Suzuki. 2022. Task-Parallel Tiled Direct Solver for Dense Symmetric Indefinite Systems. *Parallel Computing* 111 (Feb. 2022), 102900. DOI: https://doi.org/10.1016/j.parco.2022.102900

Stanimire Tomov, Jack Dongarra, and Marc Baboulin. 2010. Towards Dense Linear Algebra for Hybrid GPU Accelerated Manycore Systems. *Parallel Computing* 36, 5 (Jun. 2010), 232–240. DOI: https://doi.org/10.1016/j.parco.2009.12.005

Thomas Trogdon. 2019. On Spectral and Numerical Properties of Random Butterfly Matrices. *Applied Mathematics Letters* 95 (2019), 48–58. DOI: https://doi.org/10.1016/j.aml.2019.03.024

James H. Wilkinson. 1965. *The Algebraic Eigenvalue Problem*. Oxford University Press, London, UK.

Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. 2008. A Fast Randomized Algorithm for the Approximation of Matrices. *Applied and Computational Harmonic Analysis* 25, 3 (Nov. 2008), 335–366. DOI: https://doi.org/10.1016/j.acha.2007.12.002

Received 14 December 2023; revised 21 August 2024; accepted 25 September 2024