Using Additive Modifications in LU Factorization Instead of Pivoting

Neil Lindquist
Innovative Computing Laboratory
University of Tennessee
Knoxville, TN, USA
nlindqu1@icl.utk.edu

Piotr Luszczek
Innovative Computing Laboratory
University of Tennessee
Knoxville, TN, USA
luszczek@icl.utk.edu

Jack Dongarra
Innovative Computing Laboratory
University of Tennessee
Knoxville, TN, USA
dongarra@icl.utk.edu

ABSTRACT

Direct solvers for dense systems of linear equations commonly use partial pivoting to ensure numerical stability. However, pivoting can introduce significant performance overheads, such as synchronization and data movement, particularly on distributed systems. To improve the performance of these solvers, we present an alternative to pivoting in which numerical stability is obtained through additive updates. We implemented this approach using SLATE, a GPU-accelerated numerical linear algebra library, and evaluated it on the Summit supercomputer. Our approach provides better performance (up to 5-fold speedup) than Gaussian elimination with partial pivoting for comparable accuracy on most of the tested matrices. It also provides better accuracy (up to 15 more digits) than Gaussian elimination with no pivoting for comparable performance.

CCS CONCEPTS

• **Mathematics of computing** → **Mathematical software performance**; **Computations on matrices**; • **Computing methodologies** → *Distributed algorithms.*

KEYWORDS

LU factorization, linear algebra, communication avoidance

ACM Reference Format:
Neil Lindquist, Piotr Luszczek, and Jack Dongarra. 2023. Using Additive Modifications in LU Factorization Instead of Pivoting. In *2023 International Conference on Supercomputing (ICS '23), June 21–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3577193.3593731

1 INTRODUCTION

Solving large, dense, non-symmetric systems of linear equations is a key step in many applications [5, 20]. Gaussian elimination with partial pivoting (GEPP) is commonly used to solve these systems and provides robust numerical accuracy for almost all classes of matrices. However, partial pivoting can introduce significant overheads, including synchronizations, latency-bound pivot searches, and exchanging rows in memory. The row exchanges also must be interleaved with the Schur-complement updates, reducing the available parallelism. Finally, these overheads are made worse by the growing performance gap between arithmetic and data movement [18]. Gaussian elimination with no pivoting (GENP) can achieve significantly higher performance by avoiding these overheads but cannot accurately solve many types of systems [28]. We consider an alternative to pivoting based on low rank, additive modifications of the diagonal submatrices: we call it block elimination with additive modifications (BEAM). The goal of this approach is to incur lower overheads than GEPP while providing better numerical stability than GENP. Related modifications have been previously considered for sparse matrices [21, 26]. But, we consider dense matrices, and our proposed algorithm modifies entire blocks of the matrix instead of targeting just individual elements.

To motivate the additive strategy, consider GENP. Factoring the kth diagonal, $A[k,k]$, updates each $A[i,j]$ in the trailing matrix by

$$A[i,j] \leftarrow A[i,j] - A[i,k]\,A[k,k]^{-1}\,A[k,j] \quad (k < i, j \leq n).$$

Thus, small $A[k,k]$ entries can result in significant element growth, which in turn can lead to a large backward error [17]. Diagonal blocks with small singular values behave analogously. To prevent this growth, we propose monitoring the singular values of the diagonal blocks and modifying those having values below a predefined tolerance. These modifications give rise to a perturbed system with better numerical properties than the original one. The perturbation can then be corrected collectively with the Woodbury formula[1] [16, 29] or iterative refinement. Consequently, our work makes the following contributions:

- We propose a general scheme for *additive modifications* in the LU factorization of dense systems.
- We propose choosing additive modifications *in batches* based on diagonal blocks of the matrix instead of individual entries.
- We prove practical bounds on key condition numbers that determine the overall numerical stability of our method.
- We test the accuracy and performance for essential matrix types at scale on the Summit supercomputer with multiple GPU accelerators per node.

2 RELATED WORK

Using element-wise modifications instead of pivoting was first suggested by Stewart for sparse matrices [26]. However, preserving the

[1]This formula has a variety of names, including the Bartlett-Sherman-Morrison-Woodbury formula and the Sherman-Morrison-Woodbury formula. Hagar's expository paper provides a history of the formula and its repeated discovery [16].

sparsity limited the modifications to only a single diagonal element at a time. Additionally, the adjustments were combined into the factorization in a recursive manner that limits the available parallelism. Unfortunately, this approach has seen limited use outside the optimization community [31]. There are a few areas where BEAM differs from past efforts. First, our new algorithm enhances *dense* LU factorization which is significantly more compute-intensive than the sparse equivalent. Furthermore, dense problems are unconstrained by sparsity issues. Next, the modifications are chosen based on entire diagonal blocks instead of individual diagonal elements, giving us a greater opportunity for exploiting non-local numerical properties. Finally, all modifications are corrected simultaneously at the end of the factorization. This increases the arithmetic intensity when applying the Woodbury formula, which improves the performance on modern hardware.

An interesting variant of the additive approach is to correct the perturbations by adding extra rows and columns to the matrix [3]. That is, the system $Ax = b$ is replaced with

$$\begin{bmatrix} A + F_1F_2 & F_1 \\ F_2 & I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

where F_1F_2 modifies the appropriate diagonal elements. This variant is closely related to the derivation of the Woodbury formula via the Schur complement [16]. Unfortunately, this idea has not been explored beyond avoiding fill-in for sparse, symmetric positive definite matrices with a few dense rows. We used the Woodbury formula to apply the correction instead of this approach to simplify matrix allocation.

A final approach is to apply modifications without directly correcting them, relying instead on iterative refinement [21]. While iterative refinement based on stationary schemes can recover minor errors, large errors can slow or even prevent convergence, especially for ill-conditioned matrices. Thus, there is a trade-off in the factorization's backward error between the direct perturbations to the matrix and the error induced by element growth. Using the Woodbury formula directly addresses the perturbations, which helps alleviate this conflict. Therefore, we tested the additive approach both with and without the Woodbury formula. As before, the prior work is related to ours, but our approach chooses modifications based on diagonal blocks and is applied to dense matrices unencumbered by sparsity considerations.

Beside additive modifications, there have been various other approaches to remove or reduce the cost of pivoting. One such strategy is to replace pivoting with randomized preprocessing, which allows using an optimized GENP code for the factorization at the cost of preprocessing the matrix [24, 25]. Recent work has demonstrated significant speedups for many types of matrices; unfortunately, these approaches can fail in some cases [23]. Thus, it is valuable to investigate algorithms that may be successful for linear systems where randomization is insufficient. Furthermore, using BEAM in combination with randomized preprocessing is likely to be more robust than either of the methods separately.

Another pivoting-replacement strategy is a hybrid of the LU and QR factorizations [9]. This algorithm attempts to factor each block column with GENP. It then tests the stability and, if unstable, re-factors the block column with the unequivocally stable QR factorization. Thus, in the best case, the factorization progresses as per GENP but provides more robust behavior when GENP struggles. Unfortunately, for task-parallel implementations testing each block column for stability and the occasional QR factorization results in similar parallel dependencies to GEPP, which reduces the available parallelism. This hybrid method and our algorithm are similar in their optimism about GENP, but they differ in the mechanism by which they recover stability in problematic cases.

Finally, there have also been efforts to reduce the cost of pivoting without completely removing it. The most well-known approach is *tournament pivoting*, which computes pivots block-wise to avoid synchronizing for each column [14]. Another strategy is to relabel the rows without exchanging them in memory, although swapping rows may still be needed for load balancing [11]. The recent COnfLUX algorithm goes further by combining these strategies [19]. Finally, a recent proposal, *threshold pivoting*, tries to reduce data movement by allowing the selection of pivots smaller than the column's maximum [22]. Unfortunately, these approaches still incur significant pivoting overheads and reduce the available parallelism since the exchanged rows are unknown until runtime.

3 ADDITIVE MODIFICATIONS ALGORITHM

The core idea of our approach is to apply additive modifications during the factorization when small entries occur on the diagonal instead of exchanging rows. A straightforward way to do this is to perform the classic non-pivoted LU factorization and modify diagonal entries whenever they dip below a preset tolerance. However, this results in a myopic view of the matrix; the issues with one diagonal element can often be fixed using just the next row, for example in a matrix where the leading 2-by-2 diagonal submatrix is $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$. Thus, we use a block LU factorization where the diagonal blocks are factored with the singular value decomposition (SVD). Then, we modify the singular values that are too small. However, the SVD requires significantly more computation than an LU decomposition: $21n^3$ operations instead of $2/3n^3$—a 30× difference [12]. This limits the block size, n_b, that can be used without introducing significant overhead. Other rank-revealing factorizations, such as QR with column pivoting, are cheaper; however, the SVD is a more robust factorization, which helps us focus on the effects of the overall block-wise factorization and the additive modifications. Note that by using the SVD in this way, the final decomposition is not a regular LU factorization (unless $n_b = 1$) but a decomposition into lower and upper *block*-triangular matrices.

Additive modifications commute naturally with the preceding Schur complement updates of the block LU factorization via the commutativity of matrix addition. Hence, the modified LU factors are equivalent to the factors produced by applying all modifications before beginning the factorization (if we ignore the effects of numerical round-off). This is analogous to representing row pivoting as pre-multiplication by a permutation matrix P: $\widetilde{A} \equiv PA$. Thus, our proposed method factorizes A into

$$\widetilde{L}\widetilde{R} = \widetilde{A} \equiv A + M_U M_\Sigma M_V^T \tag{1}$$

where $\widetilde{L}$ and $\widetilde{R}$ are lower and upper block-triangular matrices, respectively, while M_Σ is a diagonal matrix containing the modifications. Note that we denote the upper block-triangular factor as $\widetilde{R}$ ("right") instead of the usual $\widetilde{U}$ ("upper") to avoid confusion

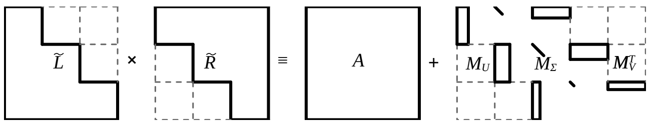

Figure 1: 3×3 **block structure of BEAM factorization.**

with the U factor of the SVD. The columns of M_U and M_V are the left and right singular vectors corresponding to the modifications in M_Σ and padded with zeros to match the size of A. Thus, M_U and M_V are tall-and-narrow matrices whose columns are a subset of a block-diagonal matrix. Figure 1 visualizes these sub-matrix structures.

Because of the perturbations, the factored matrix $\widetilde{A}^{-1}$ often only provides the solution to a nearby system, so a correction is needed to obtain the solution to the original system. We considered two approaches for this correction: iterative refinement and the Woodbury formula. While the former has a well established formulation, the latter can take various forms. The most general form of the Woodbury formula is

$$(A - BCD)^{-1} = A^{-1} + A^{-1}B(C^{-1} - DA^{-1}B)^{-1}DA^{-1}, \qquad (2)$$

although a simplified form is often used where $C \equiv I$ [16]. The term $C^{-1} - DA^{-1}B$ is called a *capacitance matrix*, and its inverse is the centerpiece of the Woodbury formula. Here, we formulate the correction as

$$(A-M_U M_\Sigma M_V^T)^{-1} = A^{-1}+A^{-1}M_U(I-M_\Sigma M_V^T A^{-1} M_U)^{-1} M_\Sigma M_V^T A^{-1}$$

instead of the more obvious

$$(A - M_U M_\Sigma M_V^T)^{-1} = A^{-1} + A^{-1}M_U(M_\Sigma^{-1} - M_V^T A^{-1} M_U)^{-1} M_V^T A^{-1}$$

to avoid the need to invert the possibly ill-conditioned M_Σ and to improve the conditioning of the entire capacitance matrix. We discuss these numerical properties further in Section 4.

We outline our approach, abbreviated BEAM, in Algorithm 1. While we describe the algorithm with a fixed block size, n_b, it can easily be extended to a variable block size. In lines 6–17, we decompose the diagonal block and apply any necessary modifications. Then, lines 18–21 proceed as per a regular blocked, non-pivoted LU factorization. Finally, we compute and factor the capacitance matrix if the Woodbury formula is needed. While computing the capacitance matrix, we form and save the C_R and C_L matrices; this reduces memory accesses at the cost of a slight increase in storage unless there are numerous modifications. In spite of the factored capacitance matrix being denoted C^{-1}, the inverse should not be formed explicitly; instead, the factored form is preferable for numerical accuracy. We factor this second matrix with GEPP, but other methods could also work. The solve step simply applies the block-triangular factors and possibly the Woodbury formula.

A key advantage of Algorithm 1 comes from the fact that it has the high-level structure of a non-pivoted, block LU. Such structure provides more parallelism than partial pivoting because the panel of L and panel of R can be updated simultaneously [8, 23]. Furthermore, it allows the trailing matrix update from one iteration to overlap with the panel updates from a subsequent iteration.

To outperform GEPP, the overhead introduced by this method must be lower than that of pivoting. To that end, we count the

Algorithm 1 BEAM algorithm's factor and solve steps. Subscripts for $A, \widetilde{L}, \widetilde{R}$ denote submatrices in terms of matrix blocks, and n_b denotes the block size.

```
 1: procedure FactorBEAM($A, \tau$)
 2:     $n_t \leftarrow n/n_b$                              ▷ number of blocks in $A$
 3:     $m \leftarrow 0$                                    ▷ number of modifications applied
 4:     $A^{(0)} \leftarrow A$
 5:     for $k = 1 : n_t$ do
 6:         $U_k, \Sigma_k, V_k^T \leftarrow \mathsf{SVD}(A_{k,k}^{(k-1)})$
 7:         for $i = 1 : n_b$ do
 8:             if $\Sigma_k[i] \le \tau$ then               ▷ is $\sigma_i$ below tolerance $\tau$
 9:                 $m \leftarrow m + 1$                    ▷ Record modification
10:                 $M_\Sigma[m, m] \leftarrow \tau - \Sigma_k[i]$
11:                 $M_U[:, m] \leftarrow [0, U_k[:, i]^T, 0]^T$
12:                 $M_V[:, m] \leftarrow [0, V_k[:, i]^T, 0]^T$
13:                 $\Sigma_k[i] \leftarrow \tau$               ▷ Apply modification
14:             end if
15:         end for
16:         $\widetilde{L}_{k,k} \leftarrow U_k$
17:         $\widetilde{R}_{k,k} \leftarrow \Sigma_k V_k^T$
18:         $\mathcal{I} \leftarrow \{k+1, k+2, \dots, n_t\}$   ▷ trailing matrix indices
19:         $\widetilde{L}_{\mathcal{I},k} \leftarrow A_{\mathcal{I},k} \widetilde{R}_{k,k}^{-1}$
20:         $\widetilde{R}_{k,\mathcal{I}} \leftarrow \widetilde{L}_{k,k}^{-1} A_{k,\mathcal{I}}$
21:         $A_{\mathcal{I},\mathcal{I}}^{(k)} \leftarrow A_{\mathcal{I},\mathcal{I}}^{(k-1)} - \widetilde{L}_{\mathcal{I},k} \widetilde{R}_{k,\mathcal{I}}$
22:     end for
23:     if $m > 0$ and using Woodbury formula then
24:         $C_R \leftarrow M_\Sigma M_V^T \widetilde{R}^{-1}$
25:         $C_L \leftarrow \widetilde{L}^{-1} M_U$
26:         $C \leftarrow I - C_R C_L$
27:         $C^{-1} \leftarrow \mathsf{FACTOR}(C)$              ▷ Using, e.g., GEPP
28:     end if
29: end procedure
30: procedure SolveBEAM($b$)
31:     $x \leftarrow \widetilde{L}^{-1} b$
32:     if $m > 0$ and using Woodbury formula then
33:         $x \leftarrow (I + C_L C^{-1} C_R) x$
34:     end if
35:     $x \leftarrow \widetilde{R}^{-1} x$
36: end procedure
```

number of arithmetic operations used in the modifications and Woodbury formula. Let n be the size of the system, m be the rank of the Woodbury correction, n_b be the size of the diagonal blocks, and ℓ_{rhs} be the number of right-hand sides. (If the Woodbury formula is not applied, $m = 0$.) Because the factors' diagonal blocks are full instead of triangular, computing the Schur complement takes an extra $n^2 n_b + O(nn_b^2)$ FLOP. Thus, BEAM without the Woodbury correction takes

$$\tfrac{2}{3}n^3 + 2n^2\ell_{rhs} + n^2 n_b + O(nn_b^2 + nn_b\ell_{rhs}) \text{ FLOP.}$$

Next, building and factoring the capacitance matrix (via GEPP) takes $2n^2m + 2nm^2 + \frac{2}{3}m^3 + O(nm)$ FLOP. Finally, the Woodbury formula requires two triangular solves and two matrix multiplies.

So, the Woodbury formula adds an extra

$$2n^2m + 2nm^2 + \tfrac{2}{3}m^3 + 4nm\ell_{rhs} + 2m^2\ell_{rhs} + O(n^2 + n\ell_{rhs}) \text{ FLOP}.$$

Hence, if $n_b, m \ll n$, the arithmetic overhead compared to GENP should be negligible. While this does not measure the cost of data movement, most of the added computations have high data locality, especially compared to pivoting.

4 THEORETICAL ANALYSIS

Because BEAM inverts $\widetilde{A}$ instead of A, it is crucial to understand the errors that arise when solving $\widetilde{A}x = b$ in finite-precision. First, the additive modifications guarantee that $\widetilde{A}$ and its block principal leading submatrices are non-singular, a prerequisite for the success of non-pivoted block-LU factorizations. Thus, this algorithm (without the Woodbury formula) computes a solution, $\widehat{x}$, that satisfies a nearby system,

$$(\widetilde{A} + \Delta\widetilde{A})\widehat{x} = (b + \Delta b), \tag{3}$$

with

$$\|\Delta\widetilde{A}\|_2 \le \widetilde{\eta}_2(\widehat{x})\|\widetilde{A}\|_2 \quad \text{and} \quad \|\Delta b\|_2 \le \widetilde{\eta}_2(\widehat{x})\|b\|_2; \tag{4}$$

that is, $\widetilde{\eta}_2(\widehat{x})$ is the normwise backward error for the spectral norm. Then, combining (3) with the definition of $\widetilde{A}$ from (1) gives

$$(A + M_U M_\Sigma M_V^T + \Delta\widetilde{A})\widehat{x} = (b + \Delta b).$$

Thus, the backward error of $\widehat{x}$ for the original system $Ax = b$ is

$$\eta_2(\widehat{x}) \le \max\left(\frac{\|M_U M_\Sigma M_V^T + \Delta\widetilde{A}\|_2}{\|A\|_2}, \frac{\|\Delta b\|_2}{\|b\|_2}\right) \le \hat{\tau} + \widetilde{\eta}_2(\widehat{x}). \tag{5}$$

Hence, the forward error of $\widehat{x}$ can be bounded with only $\hat{\tau}$, the backward stability of the block factorization, and the condition number of A. Importantly, $\widetilde{A}$ need not be well conditioned. Furthermore, the convergence of iterative refinement is also ensured when those three values are sufficiently small [4]. Note that (5) implies that $\hat{\tau}$ may directly contribute to the backward error when there are modifications but the Woodbury formula is not applied.

Unfortunately, an ill-conditioned $\widetilde{A}$ can still be problematic when using the Woodbury formula because its forward error directly perturbs the capacitance matrix and, thus, the correction. To help understand this condition number, we provide the following theorem, which states that if the tolerance, $\hat{\tau}$, is small relative to the reciprocal condition number of A, the conditioning of $\widetilde{A}$ will be close to that of A. Interestingly, the condition of this theorem ($\hat{\tau}\kappa_2(A) \ll 1$) appears related to the requirement implied by (5) for the solution to have any digits of accuracy ($\hat{\tau}\kappa_2(A) + \widetilde{\eta}_2\kappa_2(A) \ll 1$).

Theorem 4.1. *Let $\widetilde{A} = A + M_U M_\Sigma M_V^T$ where M_U and M_V each have orthonormal columns, and M_Σ is a diagonal matrix with positive entries of at most $\tau = \hat{\tau}\|A\|_2$. If $\hat{\tau}\kappa_2(A) < 1$, then*

$$\kappa_2(\widetilde{A}) \le \frac{\sigma_1(A) + \tau}{\sigma_n(A) - \tau} = \kappa_2(A)\frac{1 + \hat{\tau}}{1 - \hat{\tau}\kappa_2(A)}$$

where $\sigma_1(A)$ and $\sigma_n(A)$ denote the largest and smallest singular values, respectively, and $\kappa_2(A) = \sigma_1(A)/\sigma_n(A)$.

Proof. By the triangle inequality,

$$\sigma_1(\widetilde{A}) \le \sigma_1(A) + \tau \quad \text{and} \quad \sigma_n(\widetilde{A}) \ge \sigma_n(A) - \tau.$$

Suppose $\hat{\tau}\kappa_2(A) < 1$, which implies $\sigma_n(A) - \tau > 0$. Hence,

$$\kappa_2(\widetilde{A}) \le \frac{\sigma_1(A) + \tau}{\sigma_n(A) - \tau} = \frac{\sigma_1(A) + \hat{\tau}\sigma_1(A)}{\sigma_n(A) - \hat{\tau}\sigma_1(A)} = \kappa_2(A)\frac{1 + \hat{\tau}}{1 - \hat{\tau}\kappa_2(A)}. \qquad \square$$

When applying the Woodbury formula, we must also invert the capacitance matrix as per (2). Thus, its condition number is also crucial in the analysis of this method. We start by generalizing a lemma of Yip [30] to the full version of the Woodbury formula.

Lemma 4.2. *Let $\|\cdot\|_p$ be any sub-multiplicative matrix norm. We denote the condition number with respect to the Moore-Penrose pseudoinverse by $\kappa_p^+(A) = \|A\|_p\|A^+\|_p$. Suppose $\widetilde{A} = A + U\Sigma V^T$ is nonsingular with U, V having full column rank and Σ being nonsingular. Then,*

$$\begin{aligned}\kappa_p(\Sigma^{-1} - V^T\widetilde{A}^{-1}U) &\le \min\left(\kappa_p^+(U)^2, \kappa_p^+(V^T)^2\right)\kappa_p(\Sigma)\kappa_p(A\widetilde{A}^{-1}) \\ &\le \min\left(\kappa_p^+(U)^2, \kappa_p^+(V^T)^2\right)\kappa_p(\Sigma)\kappa_p(A)\kappa_p(\widetilde{A}).\end{aligned}$$

Proof. Because we must bound the norm of the capacitance matrix, we start by rewriting its expression. Multiplying $\widetilde{A} - U\Sigma V^T = A$ on the left by $\Sigma^{-1}U^+$ and on the right by $\widetilde{A}^{-1}U$ gives

$$(\Sigma^{-1} - V^T\widetilde{A}^{-1}U) = \Sigma^{-1}U^+A\widetilde{A}^{-1}U. \tag{6}$$

We next seek a similar expression for its inverse. Note that,

$$A\widetilde{A}^{-1}U = (\widetilde{A} - U\Sigma V^T)\widetilde{A}^{-1}U = U - U\Sigma V^T\widetilde{A}^{-1}U.$$

So, the columns of $A\widetilde{A}^{-1}U$ are within the column space of U. Since UU^+ is an orthogonal projector onto that space [13], we have $(UU^+)A\widetilde{A}^{-1}U = A\widetilde{A}^{-1}U$. Using this, we can verify that

$$(U^+\widetilde{A}A^{-1}U\Sigma)(\Sigma^{-1}U^+A\widetilde{A}^{-1}U) = I.$$

Combining this with (6) gives the desired inverse:

$$(\Sigma^{-1} - V^T\widetilde{A}^{-1}U)^{-1} = U^+\widetilde{A}A^{-1}U\Sigma.$$

Hence, the condition number can be bounded as

$$\begin{aligned}\kappa_p(\Sigma^{-1} - V^T\widetilde{A}^{-1}U) &= \|\Sigma^{-1}U^+A\widetilde{A}^{-1}U\|_p\|U^+\widetilde{A}A^{-1}U\Sigma\|_p \\ &\le \kappa_p^+(U)^2\kappa_p(\Sigma)\|A\widetilde{A}^{-1}\|_p\|\widetilde{A}A^{-1}\|_p.\end{aligned}$$

A similar argument shows that

$$\kappa_p(\Sigma - U\widetilde{A}^{-1}V^T) \le \kappa_p^+(V^T)^2\kappa_p(\Sigma)\|A\widetilde{A}^{-1}\|_p\|\widetilde{A}A^{-1}\|_p. \qquad \square$$

Using this lemma, the condition number for the capacitance matrix in the obvious form of the Woodbury formula is bounded by

$$\kappa_2(M_\Sigma^{-1} - M_V^T\widetilde{A}^{-1}M_U) \le \kappa_2(M_\Sigma)\kappa_2(A\widetilde{A}^{-1}).$$

As mentioned in Section 3, we instead formulate the Woodbury correction to get a tighter bound on the condition number:

$$\kappa_2(I - M_\Sigma M_V^T\widetilde{A}^{-1}M_U) \le \kappa_2(A\widetilde{A}^{-1}).$$

The conditioning of this latter matrix can be further improved, particularly for the 2-norm. The following theorem shows that as long as neither A nor $\widetilde{A}$ are ill-conditioned, the capacitance matrix will have an excellent condition number.

Theorem 4.3. *Suppose $\widetilde{A} = A + M_U M_\Sigma M_V^T$ where M_U and M_V each have orthonormal columns, and $\|M_\Sigma\|_2 = \tau$. Additionally, let $C = I - M_\Sigma M_V^T \widetilde{A}^{-1} M_U$. Then,*

$$\kappa_2(C) \le (1 + \tau\|\widetilde{A}^{-1}\|_2)(1 + \tau\|A^{-1}\|_2).$$

If $\tau = \hat{\tau}\|A\|_2$ with $\hat{\tau} < 1$, then we can simplify the bound to

$$\kappa_2(C) \le (1 + \tfrac{\hat{\tau}}{1-\hat{\tau}}\kappa_2(\widetilde{A}))(1 + \hat{\tau}\kappa_2(A)).$$

Proof. With $U = M_U$, $\Sigma = I$, and $V^T = M_\Sigma M_V^T$, Lemma 4.2 gives the bound $\kappa_2(C) \le \|A\widetilde{A}^{-1}\|_2 \|\widetilde{A}A^{-1}\|_2$. Since $A = \widetilde{A} - M_U M_\Sigma M_V^T$ and $\|M_U M_\Sigma M_V^T\|_2 = \tau$, a little algebra shows that

$$\kappa_2(C) \le (1 + \tau\|\widetilde{A}^{-1}\|_2)(1 + \tau\|A^{-1}\|_2).$$

Suppose $\tau = \hat{\tau}\|A\|_2$ and $\hat{\tau} < 1$. Then,

$$\|A\|_2 = \|\widetilde{A} - M_U M_\Sigma M_V^T\|_2 \le \|\widetilde{A}\|_2 + \hat{\tau}\|A\|_2,$$

and so $\|A\|_2 \le (1 - \hat{\tau})^{-1}\|\widetilde{A}\|_2$. Therefore,

$$\begin{aligned}\kappa_2(C) &\le (1 + \hat{\tau}\|A\|_2\|\widetilde{A}^{-1}\|_2)(1 + \hat{\tau}\|A\|_2\|A^{-1}\|_2) \\ &\le (1 + \tfrac{\hat{\tau}}{1-\hat{\tau}}\kappa_2(\widetilde{A}))(1 + \hat{\tau}\kappa_2(A)). \qquad \square\end{aligned}$$

After the backward error bound in (5) and the theorems on key condition numbers, one major concern remains: how backward stable is the factorization of $\widetilde{A}$? To our knowledge, no analysis exists for block LU that would apply to Algorithm 1. The closest is by Demmel, Higham, and Schreiber [7], but the block LU they analyzed differs from our factorization in two primary ways. First, the diagonal blocks are factored with GEPP instead of the SVD. Second, the diagonal blocks of the lower block-triangular factor are identity matrices instead of singular vectors. Under reasonable assumptions, they proved that it computes a solution, $\widehat{x}$, to $Ax = b$ such that

$$(A + \Delta A)\widehat{x} = b, \quad \|\widehat{x}\|_{\max} \le c(n)u(\|A\|_{\max} + \|\widetilde{L}\|_{\max}\|\widetilde{U}\|_{\max})$$

where $\widetilde{L}$ and $\widetilde{U}$ are the computed block-triangular factors and $c(n)$ is a constant dependent on n. Thus, we expect the method to be backward stable when $\|\widetilde{L}\|_{\max}\|\widetilde{U}\|_{\max}/\|\widetilde{A}\|_{\max}$ is small. For the general case, Demmel et al. proved that this ratio is at most

$$\frac{\|\widetilde{L}\|_{\max}\|\widetilde{U}\|_{\max}}{\|A\|_{\max}} \le n\rho_{\text{NP}}^3\kappa_{\max}(A) \tag{7}$$

where ρ_{NP} is the growth factor of A for GENP (i.e., the magnitude of the largest element that occurs in any Schur complement). While the previous analysis cannot guarantee the backward stability of BEAM's factorization, the similarity between the factorizations suggests such stability is likely. Furthermore, we expect a stronger version of (7) can be proven for BEAM since the norms of the inverses of the diagonal blocks are bounded.

5 EXPERIMENTAL RESULTS

To investigate the feasibility of this approach in terms of numerical stability, scalability, and performance, we implemented it in the Software for Linear Algebra Targeting Exascale (SLATE) library and evaluated it on the Summit supercomputer. SLATE is a dense linear algebra library that targets distributed-memory, GPU-accelerated systems [10]. Our code and results are available at https://doi.org/10.6084/m9.figshare.21982472.

Our implementation follows Algorithm 1 and uses a high-level structure based on SLATE's GENP routine [23]. However, we separated BEAM's algorithmic block size from the distribution tile size (the former being smaller than the latter). For simplicity, our code does not support an algorithmic block to be split across multiple tiles of SLATE and will truncate the last block in a tile to fit. But all our experiments align the block and tile sizes so that truncation only happens in the last tile. After the factorization is complete, the capacitance matrix is built and factored. While our theory defines τ in terms of $\|A\|_2$, this is expensive to compute in practice. So, our experiments instead used the Frobenius norm, $\tau = \hat{\tau}\|A\|_F$, which is closely related.

For performance purposes, we implemented a GPU routine for batched, block-triangular solves, using a recursive formulation similar to the MAGMA [1] and KBLAS [6] libraries. Because the diagonal blocks come from the SVD, these inverses can be realized by a matrix multiplication and sometimes a diagonal scaling. While cuBLAS's batched GEMM routine was effective for the trailing-matrix updates, its performance was lacking for small block sizes due to the subsequent copy or scale operation. For such cases, we implemented a custom routine that combined the multiplication and the copy to improve cache reuse and avoid extra kernel launch overheads. To reduce the effort in performance tuning, we used part of MAGMA's matrix-multiplication routine in our kernel.

5.1 Experimental Setup

We ran our experiments on eight nodes of the Summit supercomputer at Oak Ridge National Laboratory. This machine is based on IBM Power System AC922 nodes, each containing two 22-core IBM POWER9 CPUs and six NVIDIA Volta V100 GPUs. One core of each socket is reserved for the OS (Red Hat Enterprise Linux version 8.2). Most of the computational power comes from the GPUs, each providing 7.8 TFLOP/s, 16 GiB HBM2, and 900 GB/s memory bandwidth. Each CPU provides 540 GFLOP/s, 256 GiB DDR4 memory, and 170 GB/s memory bandwidth. NVIDIA NVLink provides a bidirectional 50 GB/s between components in a socket. A dual-rail EDR InfiniBand network with a non-blocking fat-tree topology connects the nodes.

We used a modified version of SLATE's test harness for our experiments. The tester was compiled with GCC 9.1.0, CUDA 11.0.3, IBM Spectrum MPI 10.4.0.3, IBM ESSL 6.1.0, Netlib LAPACK 3.8.0, and Netlib ScaLAPACK 2.1.0. We set the `smt1` flag and started MPI with `jsrun -n 16 -a 1 -c 21 -g 3 -b packed:21 -d packed` which allocates 16 processes, each bound to a single socket and its GPUs. For all experiments, we configured the tester with `--origin h --target d --ref n --grid 4x4 --panel-threads 20 --seed 1 --seedB 2 --matrixB randn --nrhs 1`. We also set the `--matrix`, `--dim`, and `--check` flags as appropriate for the experiment. For GEPP, we also set `--nb 768 --ib 64 --lookahead 1`. For GENP and BEAM, we set `--nb 512 --lookahead 2 --ib 64`, except for the experiment described in Section 5.3 which changed the last argument as appropriate for BEAM. This configuration gives a 2d block-cyclic distribution with a 4×4 process grid and blocks of size 512 or 768, as indicated by `--nb`. Note that SLATE's `ib` parameter corresponds to the n_b value discussed in this paper; SLATE's `nb` corresponds to the larger blocks used to distribute the matrix.

Table 1: Tested Matrices

Name	Description
`rand`	Random elements uniform on $[0, 1]$
`rands`	Random elements uniform on $[-1, 1]$
`randn`	Random elements normally distributed
`randb`	Random elements of 0 or 1
`randr`	Random elements of -1 or 1
`rand_dominant`	`rand` plus nI
`svd_geo`	Random matrix with singular values geometrically spaced from 10^{-8} to 1
`chebspec`	From MATLAB's `gallery` function
`circul`	From MATLAB's `gallery` function
`fiedler`	From MATLAB's `gallery` function
`kms`	From MATLAB's `gallery` function
`orthog`	From MATLAB's `gallery` function
`riemann`	From MATLAB's `gallery` function
`ris`	From MATLAB's `gallery` function
`zielkeNS`	Zielke's nonsymmetric matrix ($a = 1$) [32]

All tests were preceded by extra tests of size $n = 5000$ (with the otherwise identical configuration) to ensure that our results were not influenced by initialization costs. To measure the effects of system noise, we ran each performance test three times and computed the mean and 95% confidence interval. Except for `svd_geo`, the error and number of modifications were the same between the different runs. Due to minor non-determinism in SLATE's QR factorization, there is slight variability between runs in the entries of `svd_geo`. However, this variability is small and does not affect our conclusions or analysis, so we just present the error values and number of modifications from the first run.

To understand how our BEAM algorithm behaves across various linear systems, we used seven random and eight structured matrices in our tests. Table 1 describes these matrices. We choose a right-hand side with each element randomly taken from the normal distribution. The matrix generator was always seeded with 1 for the matrices and with 2 for the right-hand sides so that the test problems can be reproduced.

Accuracy was measured with the infinity-norm backward error:

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

Correspondingly, in experiments with iterative refinement, the refinement was terminated when this error was less than or equal to $\sqrt{n}$ times the unit roundoff ($\sim 3.5 \times 10^{-14}$ when $n = 10^5$) or after 30 iterations. We selected this criterion based on the accuracy of GEPP (see Table 2).

5.2 Baseline Accuracy and Performance Experiments

First, Table 2 compares the accuracy of BEAM against GEPP and GENP for varying values of tolerance $\hat{\tau}$. The matrices were of size 10^5, with a blocking factor of 64 for BEAM. The reported error is the infinity-norm backward error of (8). Most importantly, the error of BEAM with the Woodbury correction is smaller than or approximately equal to that of GENP for all but one case (`orthog` with $\hat{\tau} = 10^{-10}$). Furthermore, BEAM with Woodbury correction has a significantly smaller error than GENP for most matrices and only incurs NaN values for one matrix, `zielkeNS`. (Those NaN values resulting from growth-induced overflow.) These results demonstrate the ability of our approach to provide better numerical stability than GENP. Moreover, the error was smaller than 10^{-10} for many of the matrices. This implies that the iterative refinement should often converge quickly to double-precision accuracy [4]. While $\hat{\tau} = 10^{-6}$ leads to modifications for most matrices, only five of the fifteen matrices required more than ten modifications when $\hat{\tau} \leq 10^{-8}$ (one of which was accurately solved without any modifications when $\hat{\tau} = 10^{-10}$). This indicates that the proposed approach is likely effective for a large class of matrices. Additionally, many linear systems saw a significant improvement in accuracy compared to GENP, even without modification. Thus, even just applying an SVD factorization to invert the diagonal blocks increases the stability of a non-pivoted factorization.

The results become more nuanced when considering BEAM without Woodbury correction. For $\hat{\tau} = 10^{-10}$, the uncorrected solver behaved similarly to the corrected one. For larger tolerances, however, there is a significant gap between the two, particularly for $\hat{\tau} = 10^{-6}$. This indicates that the perturbation of the uncorrected modification becomes the dominant source of error when $\hat{\tau} \gtrsim 10^{-8}$. This aligns with both the $\hat{\tau}$ term in the normwise backward error bound of (5) and the recommended tolerance when applying scalar updates without correction [21]. In contrast, when the Woodbury correction was applied, increasing the tolerances always saw similar or better accuracies. This suggests that the error in the corrected case comes from the presence of small diagonal singular values and the resulting growth and not from applying the modifications or the Woodbury correction process.

Table 3 augments Table 2 by showing the time to solve the linear systems of equations (again with $n = 10^5$). Up to 30 steps of iterative refinement were used for BEAM but not for GEPP or GENP. To clarify where iterative refinement was unsuccessful, we marked the cases which failed to achieve convergence criterion for iterative refinement ($\eta_\infty(x) \lesssim 3.5 \times 10^{-14}$). Furthermore, we provide the number of refinement iterations, with 30 being the limit.

First, note that by using iterative refinement, our approach achieved an error of less than $2^{-53}\sqrt{n}$ for almost all cases. As above, `zielkeNS`'s failure involved excessive growth generating NaN values. For the remaining failures, BEAM produced a non-NaN solution, but iterative refinement failed to converge to full accuracy. These cases included `svd_geo`, `chebspec`, `fiedler`, and `riemann` with larger tolerances (and many modifications) but without Woodbury correction. These matrices are all ill-conditioned, which limits the ability of iterative refinement to converge when the inner solution is only moderately accurate [4]. For example, $\kappa_\infty(\texttt{fiedler}) = 2n(n-1) \approx 2 \times 10^{10}$ [27, pg. 159], so iterative refinement can only be expected to converge to full accuracy when $\eta_\infty(x) \lesssim 5 \times 10^{-11}$. Furthermore, this further supports the implication of both (5) and Theorem 4.1 that $\hat{\tau}$ should be chosen such that $\hat{\tau}\kappa_2(A) \ll 1$. Interestingly, these systems were successfully solved when using the Woodbury formula, despite the dire implication of Theorems 4.1 and 4.3 that $\hat{\tau}\kappa_2(A) \lesssim 1$ can lead to a large forward

Table 2: Accuracy Comparison of BEAM without Iterative Refinement Versus GEPP and GENP.

			$\hat{\tau} = 10^{-6}$			$\hat{\tau} = 10^{-8}$			$\hat{\tau} = 10^{-10}$		
Matrix	GEPP Error	GENP Error	# Mods.	Corr. Error	Uncorr. Error	# Mods.	Corr. Error	Uncorr. Error	# Mods.	Corr. Error	Uncorr. Error
`rand`	$2{\times}10^{-14}$	$3{\times}10^{-9}$	126	$2{\times}10^{-13}$	$2{\times}10^{-7}$	2	$2{\times}10^{-12}$	$4{\times}10^{-11}$	0	$5{\times}10^{-12}$	$5{\times}10^{-12}$
`rands`	$3{\times}10^{-14}$	$3{\times}10^{-10}$	59	$7{\times}10^{-13}$	$2{\times}10^{-7}$	0	$3{\times}10^{-12}$	$3{\times}10^{-12}$	0	$3{\times}10^{-12}$	$3{\times}10^{-12}$
`randn`	$4{\times}10^{-14}$	$3{\times}10^{-10}$	57	$7{\times}10^{-13}$	$2{\times}10^{-7}$	0	$2{\times}10^{-12}$	$2{\times}10^{-12}$	0	$2{\times}10^{-12}$	$2{\times}10^{-12}$
`randb`	$3{\times}10^{-14}$	NaN	89	$4{\times}10^{-13}$	$2{\times}10^{-7}$	0	$2{\times}10^{-12}$	$2{\times}10^{-12}$	0	$2{\times}10^{-12}$	$2{\times}10^{-12}$
`randr`	$3{\times}10^{-14}$	NaN	60	$5{\times}10^{-13}$	$1{\times}10^{-7}$	0	$1{\times}10^{-12}$	$1{\times}10^{-12}$	0	$1{\times}10^{-12}$	$1{\times}10^{-12}$
`rand_dominant`	$2{\times}10^{-14}$	$1{\times}10^{-14}$	0	$1{\times}10^{-14}$	$1{\times}10^{-14}$	0	$1{\times}10^{-14}$	$1{\times}10^{-14}$	0	$1{\times}10^{-14}$	$1{\times}10^{-14}$
`svd_geo`	$5{\times}10^{-15}$	$1{\times}10^{-10}$	47 424	$8{\times}10^{-14}$	$5{\times}10^{-7}$	9 072	$2{\times}10^{-13}$	$4{\times}10^{-9}$	127	$2{\times}10^{-12}$	$2{\times}10^{-11}$
`chebspec`	$3{\times}10^{-16}$	$8{\times}10^{-10}$	3 198	$3{\times}10^{-16}$	$5{\times}10^{-7}$	0	$2{\times}10^{-16}$	$2{\times}10^{-16}$	0	$2{\times}10^{-16}$	$2{\times}10^{-16}$
`circul`	$2{\times}10^{-17}$	$1{\times}10^{-14}$	2	$9{\times}10^{-16}$	$5{\times}10^{-7}$	0	$1{\times}10^{-15}$	$1{\times}10^{-15}$	0	$1{\times}10^{-15}$	$1{\times}10^{-15}$
`fiedler`	$2{\times}10^{-18}$	NaN	98 440	$3{\times}10^{-15}$	$7{\times}10^{-7}$	92 188	$4{\times}10^{-15}$	$5{\times}10^{-9}$	0	$4{\times}10^{-15}$	$4{\times}10^{-15}$
`kms`	$2{\times}10^{-16}$	$2{\times}10^{-16}$	0	$5{\times}10^{-16}$	$5{\times}10^{-16}$	0	$5{\times}10^{-16}$	$5{\times}10^{-16}$	0	$5{\times}10^{-16}$	$5{\times}10^{-16}$
`orthog`	$3{\times}10^{-15}$	$5{\times}10^{-5}$	47 216	$4{\times}10^{-7}$	$1{\times}10^{-6}$	21 420	$2{\times}10^{-5}$	$2{\times}10^{-5}$	1 022	$6{\times}10^{-4}$	$3{\times}10^{-4}$
`riemann`	$2{\times}10^{-14}$	$5{\times}10^{-13}$	43	$6{\times}10^{-16}$	$8{\times}10^{-7}$	0	$1{\times}10^{-14}$	$1{\times}10^{-14}$	0	$1{\times}10^{-14}$	$1{\times}10^{-14}$
`ris`	$3{\times}10^{-15}$	$1{\times}10^{-1}$	49 980	$3{\times}10^{-9}$	$3{\times}10^{-5}$	49 977	$3{\times}10^{-6}$	$2{\times}10^{-6}$	49 973	$7{\times}10^{-5}$	$5{\times}10^{-5}$
`zielkeNS`	$2{\times}10^{-19}$	NaN	1 594	NaN	NaN	1 594	NaN	NaN	1 594	NaN	NaN

Table 3: Performance Comparison of BEAM (using Iterative Refinement) Versus GEPP and GENP with 95% confidence intervals.

			$\hat{\tau} = 10^{-6}$				$\hat{\tau} = 10^{-8}$				$\hat{\tau} = 10^{-10}$			
Matrix	GEPP (s)	GENP (s)	Iter.	Corr. (s)	Iter.	Uncorr. (s)	Iter.	Corr. (s)	Iter.	Uncorr. (s)	Iter.	Corr. (s)	Iter.	Uncorr. (s)
`rand`	49.6±0.8	6.5±0.8*	1	13.7±0.8	3	10.9±0.8	1	10.7±0.8	1	10.2±0.8	1	10.2±0.8	1	10.3±0.8
`rands`	49.4±1.4	6.6±1.4*	1	12.2±1.4	3	10.7±1.4	1	10.3±1.4	1	10.3±1.4	1	10.2±1.4	1	10.2±1.4
`randn`	49.6±1.4*	6.5±1.4*	1	12.2±1.4	3	10.7±1.4	1	10.2±1.4	1	10.2±1.4	1	10.2±1.4	1	10.2±1.4
`randb`	49.6±1.1	6.5±1.1*	1	12.8±1.1	4	11.0±1.1	1	10.2±1.1	1	10.2±1.1	1	10.2±1.1	1	10.2±1.1
`randr`	50.0±2.4	6.6±2.4*	1	12.1±2.4	5	11.3±2.4	1	10.2±2.4	1	10.3±2.4	1	10.1±2.4	1	10.2±2.4
`rand_dominant`	24.9±0.2	6.5±0.2	0	10.0±0.2	0	10.0±0.2	0	10.0±0.2	0	10.0±0.2	0	10.1±0.2	0	10.0±0.2
`svd_geo`	49.1±2.6	6.6±2.6*	1	44.2±2.6	30	17.9±2.6*	1	24.9±2.6	30	17.9±2.6*	1	13.6±2.6	1	10.2±2.6
`chebspec`	31.3±0.7	6.5±0.7*	0	21.2±0.7	30	17.8±0.7*	0	9.8±0.7	0	9.9±0.7	0	9.9±0.7	0	9.9±0.7
`circul`	31.4±0.9	6.5±0.9	0	10.3±0.9	3	10.5±0.9	0	9.8±0.9	0	9.8±0.9	0	9.8±0.9	0	9.9±0.9
`fiedler`	39.1±0.6	6.6±0.6*	0	76.3±0.6	30	17.7±0.6*	0	71.9±0.6	30	17.7±0.6*	0	9.8±0.6	0	9.8±0.6
`kms`	24.0±0.4	6.6±0.4	0	9.7±0.4	0	9.8±0.4	0	9.7±0.4	0	9.8±0.4	0	9.8±0.4	0	9.8±0.4
`orthog`	50.5±1.2	6.5±1.2*	2	43.9±1.2	2	10.5±1.2	3	30.6±1.2	2	10.5±1.2	24	25.6±1.2	24	16.1±1.2
`riemann`	42.2±0.7	6.6±0.7*	0	11.6±0.7	30	17.8±0.7*	0	9.9±0.7	0	10.0±0.7	0	10.0±0.7	0	10.0±0.7
`ris`	39.0±0.8	6.6±0.8*	1	44.5±0.8	3	10.6±0.8	1	44.6±0.8	2	10.5±0.8	2	45.2±0.8	2	10.5±0.8
`zielkeNS`	38.5±0.6	6.6±0.6*	0	19.8±0.6*	0	9.8±0.6*	0	19.8±0.6*	0	9.8±0.6*	0	19.7±0.6*	0	9.8±0.6*

*Error larger than $2^{-53}\sqrt{n} \approx 3.5 \times 10^{-14}$.

error. The only other cases with a high iteration count were `orthog` and $\hat{\tau} = 10^{-10}$, with and without Woodbury correction. While this matrix is orthogonal and perfectly conditioned, the factorization is of very low quality, almost certainly due to a large growth factor.

BEAM outperformed GEPP in all cases except `fiedler` and `ris` with many modifications and the Woodbury formula. Furthermore, most cases show a large speedup, particularly when the Woodbury formula was not applied. (Although, for cases that failed to converge, the speedup is, of course, a moot point.) Unfortunately, BEAM had, at best, about two-thirds the performance of GENP. The block factorization seems to be the predominant source of errors, with iterative refinement only adding a significant overhead when many iterations are applied (cf. Section 5.3 and Table 4).

Interestingly, in cases for which it converged, BEAM without Woodbury correction outperformed the corrected version in all cases. Furthermore, when $\hat{\tau} = 10^{-10}$, BEAM without Woodbury correction converged in all but the `zielkeNS` case. Combining this observation with (5) and Table 2 suggests that the Woodbury formula is unnecessary for small tolerances. Furthermore, comparing the $\hat{\tau} = 10^{-6}$ and $\hat{\tau} = 10^{-10}$ columns of Table 3 shows that in all but one case, smaller tolerances give similar or better performance than larger tolerances. The one exception is `orthog` without the Woodbury formula, likely due to the excessive growth.

5.3 Effect of tolerance

We next investigated the tradeoff in performance and accuracy for a larger variety of tolerance values and blocking sizes on select matrices without iterative refinement. Table 4 shows the results. Matching Tables 2 and 3, the matrix sizes are all $n = 10^5$. Furthermore, the number of modifications and error in Table 4 for $\hat{\tau} = 10^{-6}, 10^{-8}, 10^{-10}$ correspond to the values in Table 2; however, the run times differ from Table 3 due to the omission of iterative refinement. While GEPP and GENP do not have the algorithmic blocking parameter n_b of BEAM, they implement cache-blocking with a similar structure and a block size of 64.

Both `rand_dominant` and `rand` matrix types saw BEAM significantly outperform GEPP for all configurations. However, as mentioned earlier, BEAM performed worse than GENP, even when no corrections were applied. Moreover, smaller block sizes performed slightly better, likely due to increased arithmetic for the SVDs and block-triangular solves. For `rand_dominant`, all configurations resulted in no modifications and the same accuracy. For `rand`, on the other hand, increasing the tolerance above 10^{-10} increased the accuracy when the Woodbury correction was applied but decreased the accuracy when it was not, with the number of modifications increasing in both cases. Given the number of modifications introduced when $\hat{\tau} \leq 10^{-6}$, a tolerance of 10^{-8} or 10^{-10} is a better choice, particularly when not applying the Woodbury correction. Finally, increasing the block size reduced the number of modifications and increased the accuracy in all but one case.

The structured matrices provided more interesting results. As in the previous tables, BEAM applied numerous modifications to the `orthog` matrix for all of the tested configurations. Increasing the blocking factor helped the accuracy, although it also increased the number of modifications. The best tradeoff between performance and accuracy seems to be for tolerances of 10^{-6} or 10^{-8} (depending on the block size) without the Woodbury correction. Unexpectedly, a smaller block size led to fewer modifications; we suspect this is due to element growth in the later diagonals. For `zielkeNS`, only $\hat{\tau} = 10^{-4}$ without the Woodbury formula produced a non-NaN solution. On the other hand, the block size had limited effect on the performance or the accuracy. For $\hat{\tau} = 10^{-4}$, all three block factor sizes resulted in the modification of about 95% of the diagonal singular values, whereas for smaller tolerance values, the number of modifications was just slightly larger than the number of blocks.

5.4 Scaling Results

Finally, we tested the performance of the different solvers as the size, n, varies for the `rand_dominant`, `rand`, and `orthog` matrices. BEAM achieved speedups ranging from 4× to almost 5× for the three matrices compared to GEPP applied to `rand`. BEAM was configured with a blocking factor size of $n_b = 64$ and a tolerance of $\hat{\tau} = 10^{-8}$. BEAM with iterative refinement ran out of GPU memory for $n = 250\,000$ due to the extra copy of the system matrix. Note that a diagonally dominant matrix, such as the `rand_dominant`, is the best-case scenario for the performance of GEPP because the selected pivots already reside on the diagonal and the memory traffic of exchanging rows is avoided (though we still perform the pivot search for each column). For the large matrices, BEAM reached 80% of GENP's performance for `rand_dominant` and `rand`, as the former required no modifications and the latter required just a few modifications. Without the Woodbury correction, BEAM also performed similarly on `orthog`. However, with the correction, the performance dropped to approximately that of the best case for GEPP. Adding iterative refinement slightly reduced the overall performance, but BEAM still outperformed the best-case scenario of GEPP by 84% to 162% on the `rand_dominant` and `rand` matrices. Without the Woodbury formula, BEAM performed almost as well on `orthog` as on `rand` with speedups of 70% to 144%. With the Woodbury formula, BEAM performed in the range of GEPP, between 40% and 112% faster than the GEPP's performance on `rand` (which is close to its performance on `orthog`, as per Table 3). These speedups for `orthog` are particularly promising because most approaches struggle to accurately outperform GEPP on this matrix, especially for large sizes [8, 22–24].

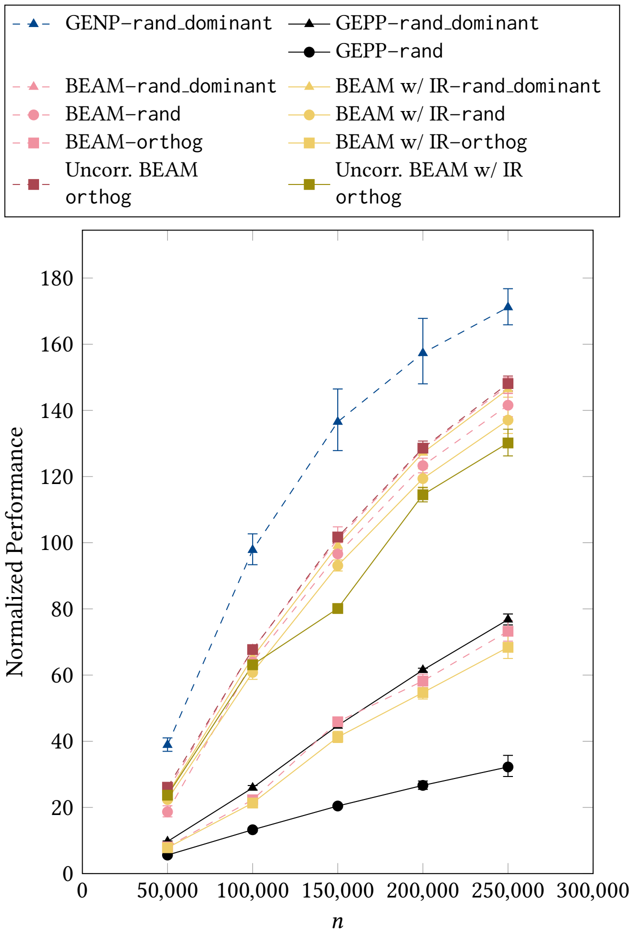

Figure 2: Performance of BEAM for three matrices compared with GEPP and GENP. The y-axis is equal to $\frac{2}{3}n^3 10^{-12}$ divided by the time in seconds; for GEPP and GENP, this is equivalent to TFLOP/s.

Table 4: Tradeoffs between performance and accuracy on select matrices for tolerance values of $\{10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}\}$ and block sizes of $\{32, 64, 128\}$ without iterative refinement.

			$n_b = 32$			$n_b = 64$			$n_b = 128$		
		Corr.	# Mods.	Time (s)	Error	# Mods.	Time (s)	Error	# Mods.	Time (s)	Error
rand_dominant	GEPP	-	-	-	-	-	25.1±0.5	2×10^{-14}	-	-	-
	GENP	-	-	-	-	-	6.7±0.4	1×10^{-14}	-	-	-
	$\hat{\tau} = 10^{-4}$	Y	0	9.8±0.2	1×10^{-14}	0	9.8±0.1	1×10^{-14}	0	10.9±0.3	1×10^{-14}
		N		9.6±0.1	1×10^{-14}		9.6±0.3	1×10^{-14}		10.8±0.4	1×10^{-14}
	$\hat{\tau} = 10^{-6}$	Y	0	9.5±0.4	1×10^{-14}	0	9.7±0.1	1×10^{-14}	0	10.9±0.2	1×10^{-14}
		N		9.5±0.2	1×10^{-14}		9.7±0.4	1×10^{-14}		10.8±0.1	1×10^{-14}
	$\hat{\tau} = 10^{-8}$	Y	0	9.5±0.2	1×10^{-14}	0	9.8±0.1	1×10^{-14}	0	10.9±0.2	1×10^{-14}
		N		9.5±0.3	1×10^{-14}		9.7±0.1	1×10^{-14}		10.9±0.2	1×10^{-14}
	$\hat{\tau} = 10^{-10}$	Y	0	9.6±0.3	1×10^{-14}	0	9.7±0.4	1×10^{-14}	0	11.0±0.3	1×10^{-14}
		N		9.4±0.2	1×10^{-14}		9.6±0.2	1×10^{-14}		10.8±0.1	1×10^{-14}
	$\hat{\tau} = 10^{-12}$	Y	0	9.4±0.4	1×10^{-14}	0	9.7±0.2	1×10^{-14}	0	10.8±0.2	1×10^{-14}
		N		9.5±0.2	1×10^{-14}		9.8±0.2	1×10^{-14}		10.8±0.2	1×10^{-14}
rand	GEPP	-	-	-	-	-	50.0±0.6	2×10^{-14}	-	-	-
	GENP	-	-	-	-	-	6.7±0.4	3×10^{-9}	-	-	-
	$\hat{\tau} = 10^{-4}$	Y	8 798	24.5±1.2	4×10^{-14}	8 397	23.7±0.9	2×10^{-14}	8 206	24.7±0.8	9×10^{-15}
		N		9.5±0.2	6×10^{-5}		9.7±0.3	5×10^{-5}		10.9±0.2	4×10^{-5}
	$\hat{\tau} = 10^{-6}$	Y	138	13.0±0.3	5×10^{-13}	126	12.9±0.3	2×10^{-13}	122	14.0±0.1	1×10^{-13}
		N		9.5±0.2	2×10^{-7}		9.7±0.3	2×10^{-7}		10.7±0.3	2×10^{-7}
	$\hat{\tau} = 10^{-8}$	Y	2	10.1±0.2	3×10^{-12}	2	10.2±0.1	2×10^{-12}	1	11.2±0.2	1×10^{-12}
		N		9.5±0.3	1×10^{-9}		9.7±0.5	4×10^{-11}		10.8±0.2	2×10^{-10}
	$\hat{\tau} = 10^{-10}$	Y	0	9.6±0.5	7×10^{-11}	0	9.8±0.2	5×10^{-12}	0	10.7±0.4	1×10^{-12}
		N		9.5±0.2	7×10^{-11}		9.8±0.2	5×10^{-12}		10.8±0.2	1×10^{-12}
	$\hat{\tau} = 10^{-12}$	Y	0	9.4±0.3	7×10^{-11}	0	9.7±0.2	5×10^{-12}	0	10.7±0.1	1×10^{-12}
		N		9.5±0.3	7×10^{-11}		9.7±0.3	5×10^{-12}		10.7±0.2	1×10^{-12}
orthog	GEPP	-	-	-	-	-	50.2±1.2	3×10^{-15}	-	-	-
	GENP	-	-	-	-	-	6.6±0.3	5×10^{-5}	-	-	-
	$\hat{\tau} = 10^{-4}$	Y	48 493	43.6±2.3	6×10^{-8}	49 065	44.0±2.5	5×10^{-11}	49 314	45.3±1.7	2×10^{-11}
		N		9.6±0.4	1×10^{-4}		9.8±0.1	1×10^{-4}		10.5±0.3	1×10^{-4}
	$\hat{\tau} = 10^{-6}$	Y	44 980	41.8±2.0	1×10^{-6}	47 216	43.1±1.7	4×10^{-7}	48 470	44.8±2.0	8×10^{-8}
		N		9.6±0.2	1×10^{-6}		9.7±0.2	1×10^{-6}		10.7±0.3	1×10^{-6}
	$\hat{\tau} = 10^{-8}$	Y	1 239	18.8±0.3	4×10^{-4}	21 420	29.6±1.1	2×10^{-5}	46 159	43.5±2.2	2×10^{-7}
		N		9.5±0.2	4×10^{-4}		9.7±0.3	2×10^{-5}		10.7±0.3	2×10^{-7}
	$\hat{\tau} = 10^{-10}$	Y	987	18.1±0.4	4×10^{-4}	1 022	18.2±0.2	6×10^{-4}	1 121	19.9±0.4	4×10^{-4}
		N		9.5±0.1	5×10^{-4}		9.7±0.2	3×10^{-4}		10.6±0.0	5×10^{-4}
	$\hat{\tau} = 10^{-12}$	Y	846	18.0±0.7	5×10^{-4}	873	18.2±0.3	2×10^{-4}	892	19.2±0.5	5×10^{-4}
		N		9.5±0.2	1×10^{-4}		9.6±0.6	3×10^{-4}		10.5±0.2	5×10^{-4}
zielkeNS	GEPP	-	-	-	-	-	39.5±0.3	2×10^{-19}	-	-	-
	GENP	-	-	-	-	-	6.7±0.3	NaN	-	-	-
	$\hat{\tau} = 10^{-4}$	Y	96 875	89.4±3.7	NaN	95 313	89.1±4.2	NaN	95 313	90.3±1.8	NaN
		N		9.3±0.5	7×10^{-5}		9.7±0.2	7×10^{-5}		10.8±0.2	7×10^{-5}
	$\hat{\tau} = 10^{-6}$	Y	3 156	21.2±0.9	NaN	1 594	19.4±0.7	NaN	813	19.1±0.3	NaN
		N		9.3±0.2	NaN		9.6±0.1	NaN		10.7±0.3	NaN
	$\hat{\tau} = 10^{-8}$	Y	3 156	21.0±0.9	NaN	1 594	19.5±0.7	NaN	813	19.1±0.3	NaN
		N		9.3±0.4	NaN		9.5±0.3	NaN		10.7±0.3	NaN
	$\hat{\tau} = 10^{-10}$	Y	3 156	20.9±0.7	NaN	1 594	19.4±0.4	NaN	813	19.2±0.0	NaN
		N		9.4±0.5	NaN		9.6±0.1	NaN		10.6±0.2	NaN
	$\hat{\tau} = 10^{-12}$	Y	3 156	20.9±0.5	NaN	1 594	19.5±0.7	NaN	813	19.2±0.3	NaN
		N		9.3±0.6	NaN		9.5±0.2	NaN		10.6±0.3	NaN

6 CONCLUSIONS

We proposed using additive modifications in Gaussian elimination for dense matrices to avoid the performance overheads associated with partial pivoting while retaining the numerical stability achieved in classic implementations of LU factorization. As a result, we recorded speedups reaching as high as 5× against the GEPP implementation on a GPU-accelerated supercomputer. Our method of additive modifications for dense matrices (unlike single-element modifications occasionally used for sparse matrices) factors the diagonal blocks with the SVD to reduce the number of modifications required. We experimentally established that BEAM provides better performance for comparable accuracy to GEPP and better accuracy for comparable performance to GENP for the majority of our test matrices. Furthermore, by testing this approach both with and without the Woodbury formula, we find that (when using iterative refinement) omitting the Woodbury correction often provides a better time-to-solution than when applying the correction.

6.1 Parameter Selection

The success of BEAM depends heavily on both the tolerance and whether to apply the Woodbury formula. The block size also matters but, based on Table 4, to a lesser extent; we suggest starting with the size of cache-blocking for non-pivoted LU or slightly larger. Below, we analyze in detail considerations for choosing the threshold and whether to use the Woodbury formula. But as a starting point, we suggest $\hat{\tau} = \min(0.5\kappa_2(A), 10^{-8})$ and no Woodbury formula. For most applications, however, various configurations should be tested on the linear systems produced by representative domain problems.

For the Woodbury formula, recall that in Table 3 and Figure 2, the corrected solver never outperformed the corresponding uncorrected one. However, a few ill-conditioned cases failed to converge without the Woodbury formula, while all cases converged when using the Woodbury formula (except for `zielkeNS`, which overflowed for both). Furthermore, as noted before, Table 3 indicates that using the Woodbury formula can enable convergence when $\hat{\tau}\kappa_2(A) \geq 1$. Thus, the Woodbury formula appears to be preferable for ill-conditioned matrices. And, iterative refinement already measures the quality of the factorization. So, we suggest initially skipping the Woodbury formula. Then if iterative refinement fails to converge within, e.g., five iterations, use the Woodbury formula in subsequent iterations.

For selecting $\hat{\tau}$, we first wish to draw attention to the importance of the inequality $\hat{\tau}\kappa_2(A) \ll 1$. For BEAM without a Woodbury correction, (5) implies that this inequality is a prerequisite to proving that the solution has at least one digit of accuracy and that iterative refinement can converge to full backward accuracy. For BEAM with a Woodbury correction, this is necessary to show that $\widetilde{A}$ is well conditioned using Theorem 4.1. Finally, our experimental results demonstrated that violating this inequality can lead to a failure of BEAM without the Woodbury formula. Although, our experimental results also failed to show a similar result when the Woodbury formula was applied. Thus, there may be a subtle interaction between the perturbations and the resulting Woodbury correction that leads to better stability than the existing analysis suggests.

Beyond ensuring $\hat{\tau}\kappa_2(A) \ll 1$, there are a few relative concerns in the selection of $\hat{\tau}$. First, consider the omission of the Woodbury correction. Recall that in Table 3, the smallest tolerance (i.e., 10^{-10}) outperformed the largest tolerance (i.e., 10^{-6}) in all but one case. Furthermore, the added perturbations become overwhelmed by the roundoff perturbations when $10^{-10} \lesssim \hat{\tau} \lesssim 10^{-8}$ (depending on the matrix). Thus, a small tolerance, such as the square root of unit roundoff, is preferable. Next, consider the inclusion of the Woodbury correction. Here the number of modifications becomes relevant in addition to their magnitude. Unfortunately, we know of no way to determine a priori the number of modifications that will result from a given tolerance. However, Table 3 suggests that, like in the uncorrected case, smaller tolerances usually result in better performance. Thus, we recommend starting with a similar tolerance to the uncorrected case.

6.2 Future Research Directions

Because of the novelty of this approach, there are several areas in which it can be improved. First, the theoretical analysis could be significantly extended, particularly for growth factors and numerical errors in both the worst and the average cases. Second, the experimental accuracy results indicate that matrices with many modifications are sensitive to a small tolerance. Thus, it would be beneficial to explore using a more dynamic policy that increases the tolerance if too many modifications are made. Relatedly, the overhead of applying correction is only significant when many modifications are applied. Thus, a scheme where only a subset of modifications (e.g., the largest twenty) are corrected may better exploit the tradeoffs between BEAM with and without the Woodbury correction. Next, the SVD will be expensive for large block sizes. Replacing it with a cheaper factorization, such as pivoted QR, may allow for increasing the block size without impacting performance. Finally, some previous efforts involving element-wise modifications have included a permutation to help place large elements on the diagonal before starting the factorization [21]. Such permutations may improve the reliability of BEAM (and other pivot-avoiding methods) since many of the troublesome matrices have leading blocks with small norms.

Another area of potential future work is in applying BEAM to matrices with exploitable structure since it provides numerical stability without destroying the block structure. First, solving symmetric-indefinite matrices requires either an LU factorization (which cannot exploit symmetry) or symmetric pivoting (which incurs complicated data-access patterns that are prohibitive on modern memory hierarchies). So, BEAM could be used to exploit the matrix's symmetry while avoiding the complexity of symmetric pivoting. This would likely take the form of a block LDLT factorization, with a symmetric-eigenvalue decomposition replacing BEAM's SVD. Second, while the approach of element-wise modifications has been previously explored for sparse matrices [21, 26], our proposal of choosing modifications based on diagonal blocks has not. This would be particularly useful for block-sparse matrices or multi-frontal methods where the diagonal blocks are already dense and no extra fill-in would be incurred. Finally, there are several matrix formats designed to exploit low-rank structures within dense matrices [2, 15]. Unfortunately, pivoting is quite restricted in such formats, which limits the matrices to which they can be safely applied. Thus, BEAM may improve the numerical stability of factorizations using such bespoke formats.

ACKNOWLEDGMENTS

This material is based upon work supported 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. Additionally, 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.

REFERENCES

[1] Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. 2017. Novel HPC Techniques to Batch Execution of Many Variable Size BLAS Computations on GPUs. In *Proceedings of the International Conference on Supercomputing (ICS '17)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3079079.3079103

[2] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo A. Mary. 2019. Bridging the Gap between Flat and Hierarchical Low-Rank Matrix Formats: The Multilevel Block Low-Rank Format. *SIAM Journal on Scientific Computing* 41, 3 (Jan. 2019), A1414–A1442. https://doi.org/10.1137/18M1182760

[3] Knud D. Andersen. 1996. A Modified Schur-complement Method for Handling Dense Columns in Interior-Point Methods for Linear Programming. *ACM Trans. Math. Software* 22, 3 (Sept. 1996), 348–356. https://doi.org/10.1145/232826.232937

[4] 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. https://doi.org/10.1137/17M1140819

[5] Chang Zhai, Yingyu Liu, Shugang Jiang, Zhongchao Lin, and Xunwang Zhao. 2020. Integrated Simulation and Analysis of Super Large Slotted Waveguide Array. *Applied Computational Electromagnetics Society Journal* 35, 7 (July 2020), 813–820.

[6] Ali Charara, David Keyes, and Hatem Ltaief. 2019. Batched Triangular Dense Linear Algebra Kernels for Very Small Matrix Sizes on GPUs. *ACM Trans. Math. Software* 45, 2 (May 2019), 15:1–15:28. https://doi.org/10.1145/3267101

[7] James W. Demmel, Nicholas J. Higham, and Robert S. Schreiber. 1995. Stability of Block LU Factorization. *Numerical Linear Algebra with Applications* 2, 2 (1995), 173–190. https://doi.org/10.1002/nla.1680020208

[8] 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 (2015), 1292–1309. https://doi.org/10.1002/cpe.3306

[9] 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. *J. Parallel and Distrib. Comput.* 85 (2015), 32–46. https://doi.org/10.1016/j.jpdc.2015.06.007

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

[11] George A. Geist and Charles H. Romine. 1988. LU Factorization Algorithms on Distributed-Memory Multiprocessor Architectures. *SIAM J. Sci. Statist. Comput.* 9, 4 (July 1988), 639–649. https://doi.org/10.1137/0909042

[12] Gene H. Golub and Chales F. Van Loan. 2013. *Matrix Computations* (fourth ed.). The John Hopkins University Press, Baltimore, MD, USA.

[13] T. N. E. Greville. 1966. Note on the Generalized Inverse of a Matrix Product. *SIAM Rev.* 8, 4 (Oct. 1966), 518–521. https://doi.org/10.1137/1008107

[14] Laura Grigori, James W. Demmel, and Hua Xiang. 2011. CALU: A Communication Optimal LU Factorization Algorithm. *SIAM J. Matrix Anal. Appl.* 32, 4 (Oct. 2011), 1317–1350. https://doi.org/10.1137/100788926

[15] Wolfgang Hackbusch. 2015. *Hierarchical Matrices: Algorithms and Analysis.* Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-47324-5_1

[16] William W. Hager. 1989. Updating the Inverse of a Matrix. *SIAM Rev.* 31, 2 (June 1989), 221–239. https://doi.org/10.1137/1031049

[17] Nicholas J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms* (second ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. https://doi.org/10.1137/1.9780898718027

[18] Awais Khan, Hyogi Sim, Sudharshan S. Vazhkudai, Ali R. Butt, and Youngjae Kim. 2021. An Analysis of System Balance and Architectural Trends Based on Top500 Supercomputers. In *The International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2021)*. Association for Computing Machinery, New York, NY, USA, 11–22. https://doi.org/10.1145/3432261.3432263

[19] Grzegorz Kwasniewski, Marko Kabic, Tal Ben-Nun, Alexandros Nikolaos Ziogas, Jens Eirik Saethre, André Gaillard, Timo Schneider, Maciej Besta, Anton Kozhevnikov, Joost VandeVondele, and Torsten Hoefler. 2021. On the Parallel I/O Optimality of Linear Algebra Kernels: Near-Optimal Matrix Factorizations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3458817.3476167

[20] Cornwall Lau, E. F. Jaeger, Nicola Bertelli, Lee A. Berry, David L. Green, Masanori Murakami, Jin M. Park, Robert I. Pinsker, and Ron Prater. 2018. AORSA Full Wave Calculations of Helicon Waves in DIII-D and ITER. *Nuclear Fusion* 58, 6, Article 066004 (April 2018), 13 pages. https://doi.org/10.1088/1741-4326/aab96d

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

[22] 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, USA, 34–42. https://doi.org/10.1109/ScalAH56622.2022.00010

[23] 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, USA, 35–43. https://doi.org/10.1109/ScalA51936.2020.00010

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

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

[26] Gilbert W Stewart. 1974. Modifying Pivot Elements in Gaussian Elimination. *Math. Comp.* 28, 126 (1974), 537–542. https://doi.org/10.1090/S0025-5718-1974-0343559-8

[27] John Todd. 1977. *Basic Numerical Mathematics.* Birkhäuser, Basel. https://doi.org/10.1007/978-3-0348-7286-7

[28] Lloyd N. Trefethen and Robert S. Schreiber. 1990. Average-Case Stability of Gaussian Elimination. *SIAM J. Matrix Anal. Appl.* 11, 3 (July 1990), 335–360. https://doi.org/10.1137/0611023

[29] Max A. Woodbury. 1950. *Inverting Modified Matrices.* Memorandum Report, Vol. 42. Statistical Research Group, Princeton, NJ.

[30] E. L. Yip. 1986. A Note on the Stability of Solving a Rank-p Modification of a Linear System by the Sherman-Morrison-Woodbury Formula. *SIAM J. Sci. Statist. Comput.* 7, 2 (April 1986), 507–513. https://doi.org/10.1137/0907034

[31] Hong Zheng and Jianlin Li. 2007. A Practical Solution for KKT Systems. *Numerical Algorithms* 46, 2 (Oct. 2007), 105–119. https://doi.org/10.1007/s11075-007-9129-8

[32] G. Zielke. 1974. Testmatrizen mit maximaler Konditionszahl. *Computing* 13, 1 (March 1974), 33–54. https://doi.org/10.1007/BF02268390