Threshold Pivoting for Dense LU Factorization

Neil Lindquist*, Mark Gates*, Piotr Luszczek*, and Jack Dongarra*
*Innovative Computing Laboratory
University of Tennessee
Knoxville, TN, USA
{nlindqu1,mgates3,luszczek,dongarra}@icl.utk.edu

***Abstract*— LU factorization is a key approach for solving large, dense systems of linear equations. Partial row pivoting is commonly used to ensure numerical stability; however, the data movement needed for the row interchanges can reduce performance. To improve this, we propose using threshold pivoting to find pivots almost as good as those selected by partial pivoting but that result in less data movement. Our theoretical analysis bounds the element growth similarly to partial pivoting; however, it also shows that the growth of threshold pivoting for a given matrix cannot be bounded by that of partial pivoting and vice versa. Additionally, we experimentally tested the approach on the Summit supercomputer. Threshold pivoting improved performance by up to 32% without a significant effect on accuracy. For a more aggressive configuration with up to one digit of accuracy lost, the improvement was as high as 44%.**

I. Introduction

LU factorization is a key approach for solving large, dense, non-symmetric or symmetric-indefinite systems of linear equations that arise in various scientific and engineering applications. In general, pivoting is necessary for numerical stability. Usually, partial pivoting is used, whereby rows are swapped at each step to ensure that the diagonal entry is at least as large in magnitude as the entries below it. Unfortunately, arithmetic is significantly cheaper than data movement in modern supercomputers [1]. This makes exchanging rows relatively expensive, especially in distributed settings. To address this, we suggest a modification to partial pivoting that reduces the associated data movement while using pivots almost as large.

Before factoring the ith column, partial pivoting ensures that

$$|A[i,i]| \geq |A[j,i]| \qquad j = i, \ldots, n \tag{1}$$

by conditionally swapping the ith row with one below it. Threshold pivoting relaxes this constraint to

$$|A[i,i]| \geq \tau |A[j,i]| \qquad j = i, \ldots, n \tag{2}$$

where $0 \leq \tau \leq 1$ is a fixed parameter. This relaxation allows the selection of smaller pivot elements that are otherwise preferable. When $\tau = 1$, threshold pivoting is equivalent to partial pivoting. On the other hand, when $\tau = 0$, no numerical pivoting is applied. Threshold pivoting is common in sparse factorizations, primarily to avoid fill-in [2]. We propose using it in dense factorizations to reduce data movement.

II. Related Work

Using threshold pivoting in dense factorization has received limited attention to date. The subject was primarily investigated by Malard in 1991 [3], followed by Hoffman and Potma a few years later [4]. Both works investigated the use of threshold pivoting to reduce inter-process communication in LU factorization on distributed systems and experimentally demonstrated both limited loss of accuracy and significant performance improvements. Unfortunately, both works were limited to $\tau \geq 0.1$ and random matrices of size $n \leq 4096$. Malard also tested dynamic pivoting, which changes the matrix distribution instead of exchanging rows between processes [5]. Dynamic pivoting still must exchange rows for load-balancing, which Malard unsuccessfully tried to improve using threshold pivoting. We extend these works in a few ways. First, we expand the algorithm to also reduce intra-process communication. Second, we show that the element growth, and thus backward error, of threshold pivoting cannot be predicted by that of partial pivoting on the same matrix. Third, our experiments for both performance and accuracy include a wider variety of matrices of much larger size, with a broader range of thresholds, on modern GPU-accelerated hardware.

The only other known uses of threshold pivoting for dense matrices are brief tests of either element growth [6] or accuracy [7]. Additionally, those experiments do not consider performance or matrices larger than $n = 2048$.

In contrast to dense factorizations, threshold pivoting is heavily used in sparse factorizations [2]. The cost of a sparse factorization directly depends on the number of nonzero entries. Thus, it can be beneficial to select pivots causing less fill-in even if (1) is not satisfied. The literature for sparse factorizations includes a wide range of threshold recommendations from 4^{-1} to 10^{-8}; however, a threshold of 10^{-1} or 10^{-2} is commonly recommended as a general-purpose guideline [2], [8]. However, experiences from sparse solvers are not guaranteed to carry over to dense solvers. First, in sparse solvers, threshold pivoting is used almost entirely to improve the management of the nonzero structure. In dense solvers, on the other hand, the nonzero structure is always the same, but reducing row-swap overheads becomes the priority. Second, the solvers have significantly different performance profiles, with dense solvers having high arithmetic intensity for most of their kernels. Finally, sparse solvers are likely to have fewer rows that satisfy (2) since most elements are 0.

Finally, other work has developed communication-avoiding variants of LU factorization. Most notably, tournament pivoting computes an entire block of pivots in a single tree reduction [9]. Interestingly, experimental results indicate that for problems of size up to 8192, tournament pivoting sat-

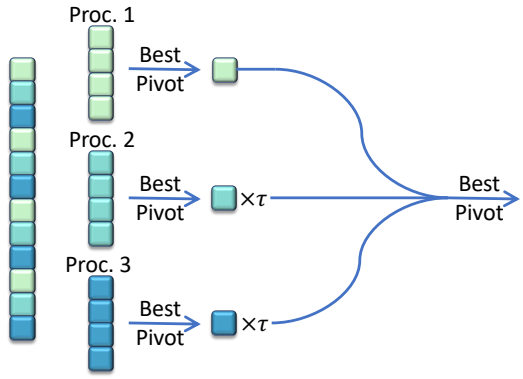

Fig. 1. Threshold pivoting for reducing inter-process data movement.

isfies (2) with $\tau \approx 0.24$ [9]. As mentioned before, data movement can also be reduced by keeping rows on their original processes [5]. These two strategies were recently combined with a 2.5D matrix distribution in the COnfLUX algorithm [10]. Alternatively, completely removing pivoting and instead randomly preconditioning the matrix before factorization has shown to be effective [11], [12]. Compared to these approaches, threshold pivoting requires less programming effort and allows control over the deviation from partial pivoting. However, it also retains some downsides of partial pivoting, such as needing a distributed reduction for each column.

III. Reducing Data Movement with Threshold Pivoting

In a dense factorization, the matrix is usually distributed using a 2D block-cyclic mapping. In such distributions, the processes are organized into a $p \times q$ grid, and the matrix is divided into contiguous blocks. These blocks are cyclically assigned to processes, i.e., the (i, j) block is assigned to the $(i \bmod p,\ j \bmod q)$ process. In these mappings, the pivot belongs to the same process as the diagonal element if and only if the corresponding rows can be exchanged without communicating between processes. So, the amount of communication can be fully determined with just the distribution of the pivot column. This equivalency, and thus the following theory, also holds for other distributions; however, this analysis may not hold for certain unusual distributions.

Using threshold pivoting to reduce the inter-process communication modifies only the pivot search of partial pivoting. For clarity's sake, we call the process that owns the current diagonal element the *root process*. First, the entry with the maximum magnitude is found for each process's local portion of the column, as in partial pivoting. Then, in the global reduction, the candidates from non-root processes are reduced by a factor of τ, as shown in Figure 1. Thus, the selected pivot is the largest element from the root process if and only if that element satisfies (2). And, if the selected pivot is not from the root process, then it is the largest element globally.

Threshold pivoting can also reduce intra-process data movement by preferring the diagonal row over other rows on the root process. This extends the above procedure by adding another global reduction to check if the diagonal element already satisfies (2). These reductions can be done simultaneously in a single `MPI_Allreduce`, so the added overhead will be small [13]. Algorithm 1 shows this approach. The local panel search is unchanged from a regular code, as per Lines 3 through 6. The best local pivot candidate is recorded as ℓ, and its value is recorded as α. The threshold logic begins at Line 7. On most processes, ℓ will be the argument for both reductions. However, if $A[j, j]$ satisfies (2) for the elements of the root process, j will be its argument for the first reduction. Both reductions scale the non-root values by τ. If the first reduction yields j, then row j is selected as the pivot. Otherwise, the result of the second reduction is selected as the pivot.

Algorithm 1 Threshold Pivoting — Panel Search

```
1:  procedure PANEL_SEARCH(A, j, τ)
2:      I ← local indices of A[j:n, j]
3:      ℓ ← first entry in I; α ← |A[ℓ, j]|
4:      for i ∈ I do
5:          if |A[i, j]| > α then ℓ ← i; α ← |A[ℓ, j]|
6:      end for
7:      ℓ1 ← ℓ; ℓ2 ← ℓ
8:      if j ∈ I then                    ▷ If this is the root process
9:          if |A[j, j]| ≥ τ|A[ℓ, j]| then ℓ1 ← j
10:         α1 ← |A[ℓ1, j]|; α2 ← |A[ℓ2, j]|
11:     else
12:         α1 ← τ|A[ℓ1, j]|; α2 ← τ|A[ℓ2, j]|
13:     end if
14:     ℓ1 ← global_argmax(ℓ1, α1)
15:     ℓ2 ← global_argmax(ℓ2, α2)
16:     if ℓ1 = j then return ℓ1
17:     else return ℓ2
18: end procedure
```

In Algorithm 1, $A[j, j]$ fulfills (2) if and only if it is the result of the first reduction because of Lines 9 and 12, respectively, for elements on the root process and on the non-root processes. Similarly, the maximum element from the root process satisfies (2) if and only if it is the result of the second maximization. Therefore, the selected pivot satisfies (2) while minimizing data movement. Even though the reductions differ only if the condition in Line 9 is true, that value is known only by the root process. So, we always compute both reductions.

Finally, this approach can be further extended to deeper hierarchies, such as using the network topology or distributing a column across multiple accelerators within a process. As before, each communication layer has a corresponding maximization where the non-local entries are penalized by a multiple of τ. Then, the reductions are considered in order of cost. If any maximization gave a local entry, the smallest such entry is taken. Otherwise, the result of the final reduction is used. As before, this gives a pivot with minimal cost that satisfies (2). Additionally, this can model the communication for complex matrix distributions by grouping rows into layers based on the amount of communication required.

IV. Theoretical Bounds for Threshold Pivoting

Gaussian elimination computes a solution, $\widehat{x}$, to the equation $Ax = b$ such that

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

$$\|\Delta A\|_\infty \leq \tfrac{3nu}{1-3nu}(1 + 2(n^2 - n)\rho(A))\|A\|_\infty,$$

$$\rho(A) = \max_{i,j,k} |A^{(k)}[i,j]| / \max_{i,j} |A[i,j]|$$

where $A^{(k)}$ is the matrix after factoring k columns, $\rho(A)$ is called the *growth factor* and u is the *unit roundoff* for the floating-point format [14, Thm. 9.4, Lemma 9.6]. Because the n^3 factor is pessimistic in practice [15], we focus our theoretical analysis on the growth factor.

The growth factor of threshold pivoting can be bounded in a similar manner as the growth factor of partial pivoting. At step k, let $\alpha = \max_{ij} |A^{(k)}[i,j]|$. Then, the values computed in the kth Schur complement have a magnitude of at most $\alpha + \tau^{-1}\alpha$. Applying this recursively gives

$$\rho(A) \leq (1 + \tau^{-1})^{n-1}. \tag{3}$$

This bound is tight and can be achieved with the matrix

$$\begin{bmatrix} \tau & 0 & \dots & 0 & 1 \\ -1 & \tau & \dots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & \dots & \tau & 1 \\ -1 & -1 & \dots & -1 & 1 \end{bmatrix},$$

which is based on Wilkinson's matrix with exponential growth in partial pivoting [16]. For $\tau = 1, 0$, (3) gives tight bounds on growth for partial pivoting and not pivoting, respectively.

Ideally, for a given matrix, the growth with threshold pivoting is bounded in terms of the growth with partial pivoting. Unfortunately, this bound is exponential in the matrix size and thus not meaningfully better than (3). We demonstrate with minor variations of Wilkinson's matrix [16]. Let

$$W_{\alpha\beta} = \begin{bmatrix} 1+\alpha & 0 & \dots & 0 & 1 \\ -1 & 1 & \dots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & \dots & 1 & 1 \\ -1-\beta & -1 & \dots & -1 & 1 \end{bmatrix} \quad \text{and}$$

$$\Omega_{\alpha\beta} = \begin{bmatrix} -1-\beta & -1 & \dots & -1 & 1 \\ -1 & 1 & \dots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & \dots & 1 & 1 \\ 1+\alpha & 0 & \dots & 0 & 1 \end{bmatrix}.$$

These matrices differ only in the exchange of the first and last rows. Let $0 < \tau < 1$ and $0 < \delta < \max(\tau^{-1} - 1, 1)$. Assume the diagonal element is selected as the pivot if it satisfies (1) or (2), as appropriate. Then, apply partial pivoting and threshold pivoting to $W_{0\delta}$ and $\Omega_{\delta 0}$. For both matrices, partial pivoting will exchange the first and last rows at the first step (giving $\Omega_{0\delta}$ and $W_{\delta 0}$), while threshold pivoting will not. We show below that the growth is exponential for $W_{\delta 0}$ and $W_{0\delta}$ but constant for $\Omega_{\delta 0}$ and $\Omega_{0\delta}$. Hence, the growth of threshold pivoting is not meaningfully bounded by that of partial pivoting, and vice versa. These drastic differences likely relate to Trefethen's conjecture that exponential growth is rare for partial pivoting because such growth "correspond[s] to unstable 'modes' that are themselves somehow unstable" [17].

A. Growth of $W_{\delta 0}$ and $W_{0\delta}$

First, consider $W_{\delta 0}$ and $W_{0\delta}$. Because the upper triangular part is mostly 0 for both, δ appears only in the last column after the first Schur complement. So, the factorization will not pivot, as per Wilkinson's matrix. Because the δ in $W_{0\delta}$ occurs in the last row, it increases only the (n, n) element by δ. Thus, the growth of $W_{0\delta}$ is

$$(1 + \delta)^{-1}(2^{n-1} + \delta) \approx 2^{n-1}.$$

For $W_{\delta 0}$, all entries in the last column are $(2 + \delta)/(1 + \delta)$ after the first Schur complement. Then, the elements in the last column double at each step, as per Wilkinson's matrix. Thus, the growth of $W_{\delta 0}$ is

$$(1 + \delta)^{-2}(2 + \delta)2^{n-2} \approx 2^{n-1}.$$

B. Growth of $\Omega_{\delta 0}$

Next, consider $\Omega_{\delta 0}$. We define $\Omega_{\delta 0}^{(k)}$ to be the matrix after k steps of elimination on $\Omega_{\delta 0}$. For notational simplicity, the indices are offset by k, i.e., the lower-right most element is always $\Omega_{\delta 0}^{(k)}[n, n]$. The first row of U is trivially all -1, while the first column of L is all 1 except for the last element, which is $-1 - \delta$. Applying the first Schur complement gives

$$\Omega_{\delta 0}^{(1)} = \begin{bmatrix} 2 & 1 & \dots & 1 & 0 \\ 0 & 2 & \dots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 2 & 0 \\ -1-\delta & -1-\delta & \dots & -1-\delta & 2+\delta \end{bmatrix}.$$

For $2 \leq k < i, j \leq n$, if no further pivoting occurs,

$$\Omega_{\delta 0}^{(k)}[i,j] = \frac{\det \Omega_{\delta 0}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}]}{\det \Omega_{\delta 0}^{(1)}[2{:}k, 2{:}k]}$$

by Gantmacher's determinant formula for Gaussian Elimination [18, p. 26]. Because $\Omega_{\delta 0}^{(1)}[2{:}k, 2{:}k]$ is upper triangular, its determinant is 2^{k-1}. The bound for the numerator is separated into six cases, depending on the position of $A[i,j]$ in the matrix. The first three cases correspond to on, below, and above the diagonal in the upper triangular part. The latter three cases correspond to on, below, and above the diagonal in the last row/column.

Case 1: $i = j < n$. Since $\Omega_{\delta 0}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}]$ equals the $k + 1$ leading principal submatrix,

$$\begin{aligned} \det \Omega_{\delta 0}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}] &= \det \Omega_{\delta 0}^{(1)}[2{:}k+1, 2{:}k+1] \\ &= 2^k. \end{aligned}$$

Case 2: $i < j < n$. The last row is zero for all but the last element, which is one. Expanding it gives

$$\begin{aligned}\det \Omega_{\delta 0}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}] &= 1 \det \Omega_{\delta 0}^{(1)}[2{:}k, 2{:}k] \\ &= 2^{k-1}.\end{aligned}$$

Case 3: $j < i < n$. The ith row is zero. So,

$$\det \Omega_{\delta 0}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}] = 0.$$

Case 4: $i = j = n$. The last column is zero except for the last element, which is $2 + \delta$. Expanding it gives

$$\begin{aligned}\det \Omega_{\delta 0}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}] &= (2+\delta) \det \Omega_{\delta 0}^{(1)}[2{:}k, 2{:}k] \\ &= (2+\delta)2^{k-1}.\end{aligned}$$

Case 5: $i < j = n$. The last column is zero. So,

$$\det \Omega_{\delta 0}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}] = 0.$$

Case 6: $j < i = n$. Adding $(1+\delta)^{-1}$ times the last row to the first leaves the determinant unchanged but makes the first row equal to e_1^T. Expanding this row gives 1 times the determinant of a matrix with the same structure less the first row and column. Recursively applying this procedure gives

$$\det \Omega_{\delta 0}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}] = 1 \cdot \det[-1-\delta] = -1-\delta.$$

Thus, diagonal elements are either 2 or $2+\delta$ and off-diagonal elements are either 0, $-2^{-k+1}(1+\delta)$, or ± 1, so no pivoting will occur during the factorization. Hence, the maximum element during the process is $2+\delta$ and

$$\rho(\Omega_{\delta 0}) = \frac{2+\delta}{1+\delta} \leq 2. \qquad \square$$

C. Growth of $\Omega_{0\delta}$

Finally, consider $\Omega_{0\delta}$. Again, let $\Omega_{0\delta}^{(k)}$ be the matrix after k steps of elimination on $\Omega_{0\delta}$, and assume its indices are offset by k. Applying the first Schur complement gives

$$\Omega_{0\delta}^{(1)} = (1+\delta)^{-1} \begin{bmatrix} 2+\delta & 1 & \dots & 1 & \delta \\ -\delta & 2+\delta & \dots & 1 & \delta \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\delta & -\delta & \dots & 2+\delta & \delta \\ -1 & -1 & \dots & -1 & 2+\delta \end{bmatrix}.$$

Let $\Omega_{0\delta}^{(1*)} = (1+\delta)\Omega_{0\delta}^{(1)}$. For $2 \leq k < i, j \leq n$, if no further pivoting occurs,

$$\begin{aligned}\Omega_{0\delta}^{(k)}[i,j] &= \frac{\det \Omega_{0\delta}^{(1)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}]}{\det \Omega_{0\delta}^{(1)}[2{:}k, 2{:}k]} \qquad \text{[18, p. 26]} \\ &= \frac{\det \Omega_{0\delta}^{(1*)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}]}{(1+\delta) \det \Omega_{0\delta}^{(1*)}[2{:}k, 2{:}k]}.\end{aligned}$$

Unlike $\Omega_{\delta 0}$, there are no zeros in $\Omega_{0\delta}^{(1*)}$ that can be used to simplify the determinants. However, we can introduce zeros, without changing the determinant, by adding a scalar multiple of one row to another. The numerator is again divided into six cases, with the first case addressing the denominator.

Case 1: $i = j < n$. The sub-diagonal elements in the first column can be eliminated by adding the first row times $\chi_1 = \delta/(2+\delta)$ to the subsequent rows, giving

$$\begin{bmatrix} 2+\delta & 1 & 1 & \dots & 1 \\ 0 & 2+\delta+\gamma_1 & 1+\gamma_1 & \dots & 1+\gamma_1 \\ 0 & -\delta+\gamma_1 & 2+\delta+\gamma_1 & \dots & 1+\gamma_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -\delta+\gamma_1 & -\delta+\gamma_1 & \dots & 2+\delta+\gamma_1 \end{bmatrix}$$

where $\gamma_1 = \chi_1$. Then, the sub-diagonal elements in the second column can be eliminated by adding the second row times $\chi_2 = (\delta - \gamma_1)/(2+\delta+\gamma_1)$ to the subsequent rows, giving

$$\begin{bmatrix} 2+\delta & 1 & 1 & \dots & 1 \\ 0 & 2+\delta+\gamma_1 & 1+\gamma_1 & \dots & 1+\gamma_1 \\ 0 & 0 & 2+\delta+\gamma_2 & \dots & 1+\gamma_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & -\delta+\gamma_2 & \dots & 2+\delta+\gamma_2 \end{bmatrix}$$

where $\gamma_2 = \gamma_1 + \chi_2(1+\gamma_1)$. Continuing the procedure gives multiples defined recursively by

$$\chi_\ell = \frac{\delta - \gamma_{\ell-1}}{2+\delta+\gamma_{\ell-1}}, \quad \text{and} \quad \gamma_\ell = \gamma_{\ell-1} + \chi_\ell(1+\gamma_{\ell-1})$$

with $\chi_0 = 0$ and $\gamma_0 = 0$. Note that

$$\begin{aligned}\gamma_\ell &= \frac{\gamma_{\ell-1}(2+\delta+\gamma_{\ell-1}) + (\delta-\gamma_{\ell-1})(1+\gamma_{\ell-1})}{2+\delta+\gamma_{\ell-1}} \\ &= \frac{\delta + \gamma_{\ell-1} + 2\delta\gamma_{\ell-1}}{2+\delta+\gamma_{\ell-1}}.\end{aligned}$$

If $\gamma_{\ell-1} = 0$, then $\gamma_\ell = \delta/(2+\delta)$. And if $\gamma_{\ell-1} = \delta$, then $\gamma_\ell = \delta$. Furthermore, it is straightforward to show that the partial derivative of γ_ℓ with respect to $\gamma_{\ell-1}$ is positive. Thus, if $\gamma_{\ell-1} \in [0, \delta]$, then $\gamma_\ell \in [0, \delta]$. Hence, $\gamma_\ell \in [0, \delta]$ for all ℓ.

Thus,

$$\det \Omega_{0\delta}^{(1*)}[2{:}k \cup \{i\}, 2{:}k \cup \{j\}] = \prod_{\ell=0}^{k-1}(2+\delta+\gamma_\ell) \quad \text{and}$$

$$\det \Omega_{0\delta}^{(1*)}[2{:}k, 2{:}k] = \prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell).$$

Hence,

$$\Omega_{0\delta}^{(k)}[i,j] = \frac{\prod_{\ell=0}^{k-1}(2+\delta+\gamma_\ell)}{(1+\delta)\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)} = \frac{2+\delta+\gamma_{k+1}}{1+\delta},$$

and

$$\frac{2+\delta}{1+\delta} \leq \Omega_{0\delta}^{(k)}[i,j] \leq 2.$$

Case 2: $i < j < n$. The second case is similar to the first, except the lower-right element starts as 1 and becomes $1 + \gamma_{k-2}$ after eliminating the sub-diagonal elements. So,

$$\Omega_{0\delta}^{(k)}[i,j] = \frac{(1+\gamma_{k-2})\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)}{(1+\delta)\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)} = \frac{1+\gamma_{k-2}}{1+\delta},$$

and

$$\frac{1}{1+\delta} \leq \Omega_{0\delta}^{(k)}[i,j] \leq 1.$$

Case 3: $j < i < n$. The third case is like the first two, except the lower-right element starts as $-\delta$ and becomes $-\delta + \gamma_{k-2}$ after eliminating the sub-diagonal elements. So,

$$\Omega_{0\delta}^{(k)}[i,j] = \frac{(-\delta+\gamma_{k-2})\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)}{(1+\delta)\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)} = \frac{-\delta+\gamma_{k-2}}{1+\delta},$$

and

$$\frac{-\delta}{1+\delta} \le \Omega_{0\delta}^{(k)}[i,j] \le 0.$$

Case 4: $i = j = n$. The same process can be applied, except with the multiples for the last row being scaled by δ^{-1}. Hence,

$$\begin{aligned}\Omega_{0\delta}^{(k)}[i,j] &= \frac{(2+\delta+\delta^{-1}\gamma_{k-1})\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)}{(1+\delta)\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)} \\ &= \frac{2+\delta+\delta^{-1}\gamma_{k-1}}{1+\delta},\end{aligned}$$

and

$$\frac{2+\delta}{1+\delta} \le \Omega_{0\delta}^{(k)}[i,j] \le \frac{3+\delta}{1+\delta}$$

Case 5: $i < j = n$. The fifth case is almost identical to the second, except the last column, and thus the numerator, is scaled by δ. So,

$$\frac{\delta}{1+\delta} \le \Omega_{0\delta}^{(k)}[i,j] \le \delta.$$

Case 6: $j < i < n$. Sub-diagonal elements in all but the last row can be eliminated without changing the determinant similar to before. Because the final row is all -1, multiplying it by $1+\gamma_\ell$ and adding it to the ℓth row zeros out the upper triangular part, but leaves $1+\delta$ on the diagonal. So,

$$\begin{aligned}\Omega_{0\delta}^{(k)}[i,j] &= \frac{(1+\delta)^{k-2}(-1)}{(1+\delta)\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)} \\ &= \frac{-(1+\delta)^{k-3}}{\prod_{\ell=0}^{k-2}(2+\delta+\gamma_\ell)},\end{aligned}$$

and

$$\frac{-(1+\delta)^{k-3}}{(2+\delta)^{k-1}} \le \Omega_{0\delta}^{(k)}[i,j] \le \frac{-1}{(2+2\delta)^{k-1}}.$$

Hence, the off-diagonal elements are always strictly less than the diagonal elements, ensuring that no pivoting occurs after the first step. Therefore,

$$\rho(\Omega_{0\delta}) = \frac{(3+\delta)/(1+\delta)}{1+\delta} = \frac{3+\delta}{(1+\delta)^2} \le 3. \qquad \square$$

V. Experimental Results

We tested threshold pivoting using the SLATE (Software for Linear Algebra Targeting Exascale) library, a dense linear algebra library targeting distributed, heterogeneous systems [19]. We implemented both strategies proposed in Section III as modifications of SLATE's existing LU factorization with partial pivoting [20]. The threshold is controlled through the `options` argument, demonstrating a backward-compatible addition of threshold pivoting to an existing LU factorization code.

Ten matrices were tested, five random and five structured, all of order 225 000. The random matrices were: (1) `rand` (entries uniformly distributed on $[0,1]$), (2) `rands` (entries uniformly distributed on $[-1,1]$), (3) `randn` (entries normally distributed), (4) `randb` (entries randomly selected from $\{0,1\}$), and (5) `rand+nI` which is `rand` plus n times the identity (making it diagonally dominant). The structured matrices came from the MATLAB gallery: (6) `circul`, (7) `fiedler`, (8) `orthog`, (9) `riemann`, and (10) `ris`.

A. Experimental Setup

We tested our approach on eight nodes of the Summit supercomputer at Oak Ridge National Laboratory. Each node contains two 22-core IBM POWER9 CPUs, with one core per CPU reserved for the OS, and six NVIDIA Volta V100 GPUs. Most of the computational power comes from the GPUs, each providing $7.8\,\mathrm{TFLOP/s}$, $16\,\mathrm{GiB}$ High Bandwidth Memory (HBM2), and $900\,\mathrm{GB/s}$ memory bandwidth. Each CPU provides $540\,\mathrm{GFLOP/s}$, $256\,\mathrm{GiB}$ DDR4 memory, and $170\,\mathrm{GB/s}$ memory bandwidth. NVIDIA NVLink provides a bidirectional $50\,\mathrm{GB/s}$ between components in a socket. A dual-rail EDR InfiniBand network connect the nodes.

Our code has been released in SLATE, with this specific version available at https://zenodo.org/record/6972268. Our software stack included GCC 9.1.0, CUDA 11.0.3, IBM Spectrum MPI 10.4.0.3-20210112, IBM ESSL 6.1.0-2, Netlib LAPACK 3.8.0, Netlib ScaLAPACK 2.1.0, and PAPI 6.0.0.1 [21].

Because MPI and BLAS libraries often do initialization that is reused in subsequent calls, warm-up tests of size 5000 were run before the reported tests. Hyperthreading was disabled with the `smt1` mode. The tests were run with `jsrun -n 16 -a 1 -c 21 -g 3 -b packed:21 -d packed`, which runs 16 MPI ranks over 8 nodes, with 21 cores and 3 GPUs per MPI rank. We measured performance and accuracy with SLATE's test code; however, we scaled the accuracy in the output by n to compensate for a division by n in SLATE's test code. The flags `--origin h --target d --seed 42 --seedB 24 --ref n --check y --nb 896 --ib 32 --panel-threads 18 --lookahead 3 --grid 4x4 --dim 5000,225000` were always used; the `--matrix` and `--piv-thresh` flags were set as appropriate. The `ibmpowernv-isa-0000.System.energy11_input` event was used to measure a node's cumulative energy usage.

B. Effects on Performance and Accuracy

First, we reduced just the inter-process exchanges with the approach from Figure 1. Various thresholds were tested on `rand+nI`, `rand`, and `orthog`. We ran each test three times, measuring the time to solve a double-precision system of equations with ten right-hand sides, and summarized the result with the mean and 95% confidence interval shown in Figure 2. First, the relative times for $\tau = 1$ of `rand` and `orthog` compared to `rand+nI` (which doesn't pivot) imply that slightly over half the time is spent exchanging rows. This supports our motivation to avoid data movement. Next,

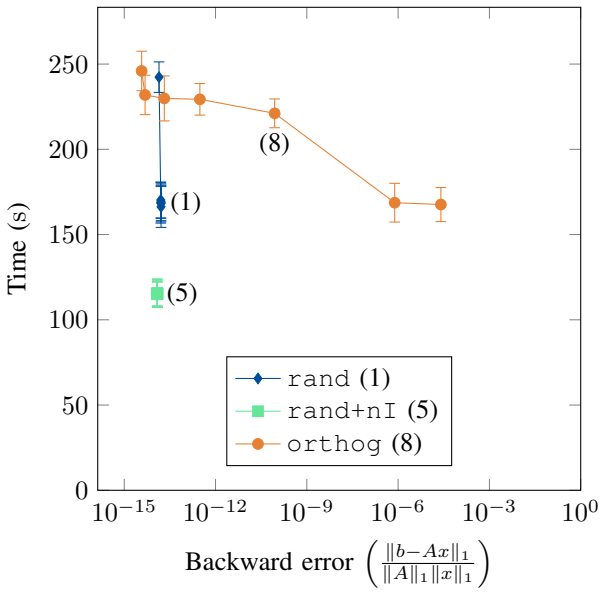

Fig. 2. Tradeoff between performance and accuracy when avoiding just inter-process row swaps. Along each line are points for thresholds of 1 (slowest), 2^{-1}, 10^{-1}, 10^{-2}, 10^{-4}, 10^{-8}, and 0 (fastest) for that matrix. For `rand`, all but the first point overlap.

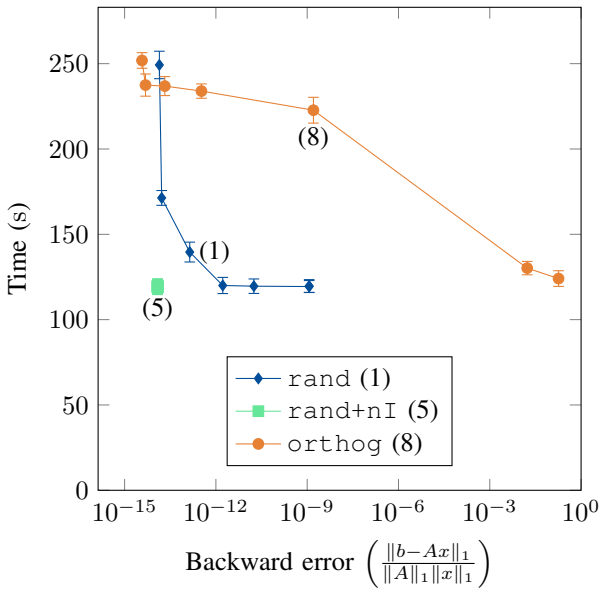

Fig. 3. Tradeoff between performance and accuracy when avoiding inter- and intra-process row swaps. Along each line are points for thresholds of 1 (slowest), 2^{-1}, 10^{-1}, 10^{-2}, 10^{-4}, 10^{-8}, and 0 (fastest) for that matrix. For `rand`, the last two points overlapped.

consider how changing τ affected the performance for each matrix. The `rand+nI` matrix was unaffected by the threshold; this is expected since partial pivoting selects diagonal pivots for diagonally dominant matrices. The `rand` matrix received a 46% speedup and a negligible effect on accuracy going from $\tau = 1$ to $\tau = 2^{-1}$, but further reductions in the threshold had little effect on either performance or accuracy. Even with $\tau = 0$, the time for the random matrix was 47% larger than for the diagonally dominant matrix; this indicates that intra-node row swaps contribute a significant overhead. The `orthog` matrix had significant reductions in accuracy as τ decreased and, for $\tau \geq 10^{-4}$, minimal increase in performance. However, the accuracy for $\tau = 2^{-1}$ was almost that of $\tau = 1$. So, for these matrices, $\tau = 2^{-1}$ does not degrade accuracy while sometimes improving performance.

Because the best performances achieved for `rand` and `orthog` were significantly below that of `rand+nI`, we repeated the previous experiment except avoiding both inter- and intra-process row swaps using Algorithm 1. Figure 3 shows the results. Avoiding both types of data movement allows the best-case performance to match that of the diagonally dominant case. However, the error increased as the tolerance was reduced. For `rand`, $\tau = 2^{-1}$ provided equal accuracy and a 31% reduction in runtime compared to partial pivoting, while $\tau = 10^{-2}$ provided equal performance and a 4-digit improvement in error compared to never exchanging rows. This provides a useful selection of τ, depending on the relative importance of performance and accuracy. On the other hand, `orthog` saw only a 6% improvement in performance compared to partial pivoting for $\tau = 2^{-1}$, although the effect on accuracy was still negligible. Even when the tolerance was as low as $\tau = 10^{-4}$, the improvement was merely 12%, but the error increased to $1.6 \cdot 10^{-9}$. This demonstrates that threshold pivoting will not consistently improve performance, although high accuracy was still obtained with large tolerance values. Compared to Figure 2, the results for $\tau = 1$ had a slight reduction in performance. However, because the corresponding confidence intervals overlap, this may stem from system noise. Furthermore, the difference was only a few percent, so even if the difference is entirely due to the increased complexity of pivot selection, the overhead is insignificant.

With the successes of $\tau = 2^{-1}$ and $\tau = 10^{-1}$ in the previous experiment, we compared the performance and accuracy of the remaining matrices for those thresholds and $\tau = 1$ (i.e., partial pivoting). For reference, we also factored `rand+nI` with an LU factorization optimized with the assumption that it will not pivot (SLATE's `gesv_nopiv`); this routine has increased parallelism and better GPU utilization but risks catastrophic numerical failure if the matrix is not diagonally dominant. Figure 4 shows the results. The added random matrices behave similarly to `rand`. However, three of the added structured matrices saw significant performance improvements with minimal reduction in accuracy, unlike `orthog`. Furthermore, two of the three (`circul` and `fiedler`) achieved performance equivalent to the diagonally dominant matrix for the smallest threshold. The last structured matrix (`ris`) was unaffected by the tested tolerance values; this likely stems from the entries being very small except along the anti-diagonal, which limits the available pivots for the moderate thresholds we tested. Interestingly, one matrix (`circul`) had higher accuracy for $\tau = 2^{-1}$ than for $\tau = 1$; this reflects the observation in Section IV that the accuracy of threshold pivoting can, on

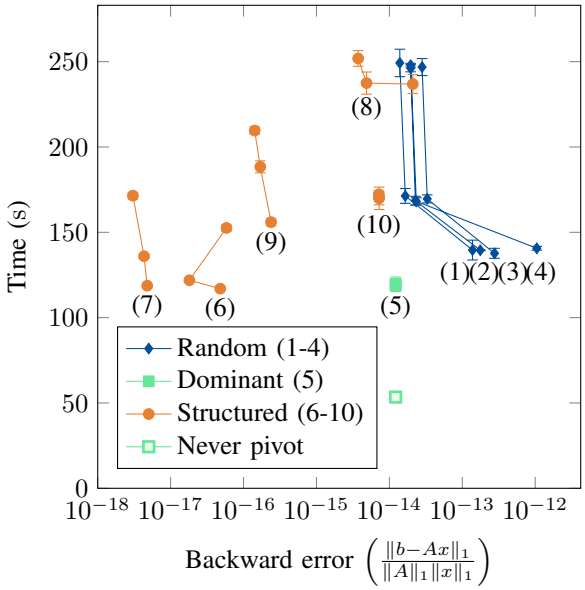

Fig. 4. Trade-off between performance and accuracy when avoiding inter- and intra-process row swaps for a variety of matrices. Along each line are points for thresholds of 1 (slowest), 2^{-1}, and 10^{-1} (fastest) for that matrix.

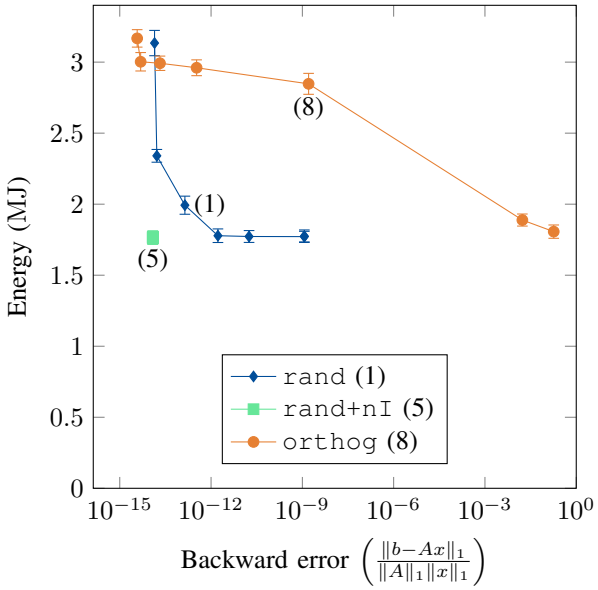

Fig. 5. Tradeoff between energy consumption and accuracy when avoiding inter- and intra-process row swaps. Along each line are points for thresholds of 1 (slowest), 2^{-1}, 10^{-1}, 10^{-2}, 10^{-4}, 10^{-8}, and 0 (fastest) for that matrix. For `rand`, the last two points overlap.

occasion, be better than that of partial pivoting.

These results indicate that a threshold of 2^{-1} or 10^{-1} is a reasonable, general-purpose default that consistently achieves an accuracy close to partial pivoting. This corresponds to the conservative recommendations from the sparse-factorization literature. However, a threshold of 10^{-2} is also commonly recommended for the general case, and recommendations for specific applications can be much smaller, e.g., 10^{-8} [8]. This reflects a lower cost to pivot in dense factorizations due to the full nonzero structure.

C. Effects on Energy Consumption

In light of the performance improvements, we tested the effect of this strategy on energy consumption. Because the runtime overhead of the intra-process exchanges was significant, only the two-layer formulation was considered for the energy consumption. We provide the results as the energy consumed to solve the system in Joules. Another metric is the "flops per watt", which is used by benchmarks such as the TOP500 [22], [23], Green500 [24], and SPEC Power [25] lists, as well as supercomputing power consumption research [26], [27]. This metric divides the performance in $\mathrm{GFLOP/s}$ by the average power usage in Watts, resulting in an efficiency metric measured in $(\mathrm{GFLOP/s})/\mathrm{W}$. However, this is inversely proportional to the total energy for a fixed n, so we provide only the latter. For the tested size, $n = 225\,000$, using 1 MJ to solve a system is equivalent to achieving $7.59\,(\mathrm{GFLOP/s})/\mathrm{W}$.

Figure 5 shows the effect of varying τ on energy usage for the `rand+nI`, `rand`, and `orthog` matrices (cf. Figure 3), while Figure 6 shows the effect of threshold pivoting on energy for all ten matrices (cf. Figure 4). The effect on energy consumption shows similar trends as runtime for both experiments, albeit with less relative improvement. This smaller

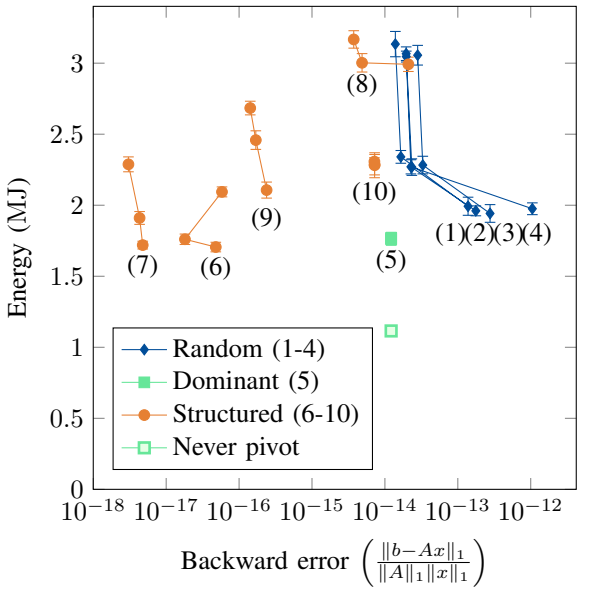

Fig. 6. Trade-off between energy and accuracy of when avoiding inter- and intra-process row swaps for a variety of matrices. Along each line are points for thresholds of 1 (slowest), 2^{-1}, and 10^{-1} (fastest) for that matrix.

improvement likely stems from the inability to remove the energy usage of tasks by parallelizing them, unlike runtime.

VI. Conclusions and Future Work

Our results demonstrate that threshold pivoting can significantly improve the performance of LU factorization without a noticeable loss of accuracy. This improvement can be realized with only minor modifications to existing partial pivoting codes, making it a worthwhile addition to libraries. Because

$\tau = 1$ gives partial pivoting, threshold pivoting with a user-controlled threshold can replace a separate partial pivoting code. This can even be done in a backward-compatible manner, as is demonstrated in our implementation. Like previous works, our results indicate $\tau = 2^{-1}$ is a reasonable starting recommendation which can be improved for specific classes of matrices and applications through experiment.

This work can be extended in multiple ways. First, as discussed in Section III, the idea could be applied to deeper communication hierarchies. Next, we used a single threshold, but interchanging local rows is cheaper than interchanging remote rows. So, using different thresholds at different levels could expose better tradeoffs between performance and accuracy. Because the results varied with the matrix, it would be valuable to test the approach on linear systems from real applications as well. In light of our results reducing intra-node pivoting and previous work on the cost of row exchanges on single node systems [28], this approach may prove to be beneficial in non-distributed settings as well. Finally, this approach may also be beneficial in other dense factorizations. Symmetric pivoting is complicated in a tiled layout [29]. But, pivots within the diagonal tile do not involve inter-tile exchanges, so preferring those entries could reduce the cost.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation Office of Advanced Cyberinfrastructure (OAC) CSE Dir. for Comp. & Info Sci. & Eng. under Grant No. 2004541, and by the Exascale Computing Project (17-SC-20-SC), 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 DE-AC05-00OR22725.

REFERENCES

[1] A. Khan, H. Sim, S. S. Vazhkudai *et al.*, "An analysis of system balance and architectural trends based on Top500 supercomputers," in *The International Conference on High Performance Computing in Asia-Pacific Region*, ser. HPC Asia 2021. New York, NY, USA: Association for Computing Machinery, Jan. 2021, pp. 11–22.

[2] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, 2nd ed. Oxford, UK: Oxford University Press, Mar. 2017.

[3] J. Malard, "Threshold pivoting for dense LU factorization on distributed memory multiprocessors," in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*. Albuquerque, NM, USA: Association for Computing Machinery, Nov. 1991, pp. 600–607.

[4] W. Hoffmann and K. Potma, "Threshold-pivoting in parallel Gaussian elimination for improved efficiency," in *Proceedings of the First Annual Conference of the Advanced School for Computing and Imaging*. Delft: Technical University Delft, 1995, pp. 63–68.

[5] G. A. Geist and C. H. Romine, "LU factorization algorithms on distributed-memory multiprocessor architectures," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 4, pp. 639–649, Jul. 1988.

[6] L. N. Trefethen and R. S. Schreiber, "Average-case stability of Gaussian elimination," *SIAM Journal on Matrix Analysis and Applications*, vol. 11, no. 3, pp. 335–360, Jul. 1990.

[7] T. Endo and K. Taura, "Highly latency tolerant Gaussian elimination," in *The 6th IEEE/ACM International Workshop on Grid Computing, 2005.* Seattle, WA, USA: IEEE Press, Nov. 2005, p. 8 pp.

[8] J. D. Hogg and J. A. Scott, "Pivoting strategies for tough sparse indefinite systems," *ACM Transactions on Mathematical Software*, vol. 40, no. 1, pp. 4:1–4:19, Oct. 2013.

[9] L. Grigori, J. W. Demmel, and H. Xiang, "CALU: A communication optimal LU factorization algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 4, pp. 1317–1350, Oct. 2011.

[10] G. Kwasniewski, M. Kabic, T. Ben-Nun *et al.*, "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*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 1–15.

[11] N. Lindquist, P. Luszczek, and J. Dongarra, "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)*. Atlanta, GA, USA: IEEE Press, Nov. 2020, pp. 35–43.

[12] V. Y. Pan and L. Zhao, "Numerically safe Gaussian elimination with no pivoting," *Linear Algebra and its Applications*, vol. 527, pp. 349–383, Aug. 2017.

[13] A. Bienz, L. Olson, and W. Gropp, "Node-aware improvements to Allreduce," in *2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI)*. Denver, CO, USA: IEEE Press, Nov. 2019, pp. 19–28.

[14] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Society for Industrial and Applied Mathematics, 2002.

[15] N. J. Higham and D. J. Higham, "Large growth factors in Gaussian elimination with pivoting," *SIAM Journal on Matrix Analysis and Applications*, vol. 10, no. 2, pp. 155–164, Apr. 1989.

[16] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*. London, UK: Oxford University Press, 1965.

[17] L. N. Trefethen, "Three mysteries of Gaussian elimination," *ACM SIGNUM Newsletter*, vol. 20, no. 4, pp. 2–5, Oct. 1985.

[18] F. R. Gantmacher, *The Theory of Matrices*. Providence, RI, USA: American Mathematical Soc., 1959, vol. 1.

[19] M. Gates, J. Kurzak, A. Charara *et al.*, "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*, ser. SC '19. Denver, CO, USA: Association for Computing Machinery, Nov. 2019, pp. 1–18.

[20] J. Kurzak, M. Gates, A. Charara *et al.*, "Linear systems solvers for distributed-memory machines with GPU accelerators," in *Euro-Par 2019: Parallel Processing*, ser. Lecture Notes in Computer Science, R. Yahyapour, Ed. Göttingen, Germany: Springer, Cham, 2019, pp. 495–506.

[21] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," in *Tools for High Performance Computing 2009*, M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, Eds. Berlin, Heidelberg: Springer, 2010, pp. 157–173.

[22] H. W. Meuer, E. Strohmaier, J. J. Dongarra, and H. D. Simon, *TOP500 Supercomputer Sites*, 32nd ed., November 2008, the report can be downloaded from http://www.netlib.org/benchmark/top500.html).

[23] H. W. Meuer, E. Strohmaier, J. J. Dongarra *et al.*, "TOP500 Supercomputing Sites," http://www.top500.org/, 2022.

[24] W.-C. Feng and K. W. Cameron, "The Green500 list: Encouraging sustainable supercomputing," *IEEE Computer*, vol. 40, no. 12, pp. 50–55, Dec. 2007.

[25] SPEC, "The SPEC power benchmark," 2008, see www.spec.org/power_ssj2008/.

[26] K. W. Cameron, R. Ge, and X. Feng, "High-performance, power-aware, distributed computing for scientific applications," *IEEE Computer*, vol. 38, no. 11, pp. 40–47, Nov. 2005.

[27] W.-C. Feng, R. Ge, and K. W. Cameron, "Power and energy profiling of scientific applications on distributed systems," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 05)*, Denver, CO, USA, Apr. 2005.

[28] A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra, "Analysis and design techniques towards high-performance and energy-efficient dense linear solvers on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2700–2712, Dec. 2018.

[29] G. Ballard, D. Becker, J. Demmel *et al.*, "Implementing a blocked Aasen's algorithm with a dynamic scheduler on multicore architectures," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. Cambridge, MA, USA: IEEE Press, May 2013, pp. 895–907.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Our experiments used modified versions of the SLATE linear algebra library. Two versions were used, one that only reduces inter-process communication and one that reduces both inter-process and intra-process communication. In both cases, the tests were run using SLATE's test code which captures accuracy, and performance. The test code was also modified to use PAPI to capture the events specified by the `PAPI_EVENTS` environmental variable (which was set to `ibmpowernv-isa-0000.System.energy11_input` in our experiments).

We ran our experiments on eight nodes of the Summit supercomputer at Oak Ridge National Laboratory. This system has two 22-core IBM POWER9 CPUs and six NVIDIA Volta V100 GPUs per node. Our software stack included GCC 9.1.0, CUDA 11.0.3, IBM Spectrum MPI 10.4.0.3-20210112, IBM ESSL 6.1.0-2, Netlib LAPACK 3.8.0, Netlib ScaLAPACK 2.1.0, and PAPI 6.0.0.1.

Hyperthreading was disabled with Summit's `smt1` mode. The tests were run with `jsrun -n 16 -a 1 -c 21 -g 3 -b packed:21 -d packed`, which runs 16 MPI ranks over 8 nodes, with 21 cores and 3 GPUs per MPI rank. We measured performance and accuracy with SLATE's test code; the accuracy was scaled by n to convert to the desired error formula. The flags `–origin h –target d –seed 42 –seedB 24 –ref n –check y –nb 896 –ib 32 –panel-threads 18 –lookahead 3 –grid 4x4 –dim 5000,225000` were always used; the `–matrix` and `–piv-thresh` flags were set as appropriate.

AUTHOR-CREATED OR MODIFIED ARTIFACTS:

Artifact 1

Persistent ID: `https://zenodo.org/record/6972268`
Artifact name: Experiment sources and results.

Reproduction of the artifact without container: The following dependencies should be installed and added to the compiler path:

(1) GCC 9.1.0
(2) CUDA 11.0.3
(3) IBM Spectrum MPI 10.4.0.3-20210112
(4) IBM ESSL 6.1.0-2
(5) Netlib LAPACK 3.8.0
(6) Netlib ScaLAPACK 2.1.0, and
(7) PAPI 6.0.0.1.

We obtained all but the last using Summit's modules. We compiled PAPI from source from tag papi-6-0-0-1-t. PAPI was configured with the `--with-component lmsensors` flag to provide the aforementioned energy event.

SLATE was then compiled using the `make -j8 test` command. A submission script for Summit is provided; although, paths and project accounts may need to be adjusted as appropriate. The script should be easily convertible to other systems. The same script can be used for both variants of SLATE. Note that, due to job length limits, the experiments are broken into two groups: the one for the 3-matrix tests, and one for the many-matrix tests.