Compressed threshold pivoting for sparse symmetric indefinite systems

JD Hogg, JA Scott

May 2013

Compressed threshold pivoting for sparse symmetric indefinite systems

J. D. Hogg and J. A. Scott

May 10, 2013

Abstract

A key technique for controlling numerical stability in sparse direct solvers is threshold partial pivoting. When selecting a pivot, the entire candidate pivot column below the diagonal must be up-to-date and must be scanned. If the factorization is parallelized across a large number of cores, communication latencies can be the dominant computational cost.

In this paper, we propose two alternative pivoting strategies for sparse symmetric indefinite matrices that significantly reduce communication by compressing the necessary data into a small matrix that can be used to select pivots. Once pivots have been chosen, they can be applied in a communication-efficient fashion.

For an $n \times p$ submatrix on P processors, we show our methods perform a factorization using $O(\log P)$ messages instead of the $O(p \log P)$ for threshold partial pivoting. The additional costs in terms of operations and communication bandwidth are relatively small.

A stability proof is given and numerical results using a range of symmetric indefinite matrices arising from practical problems are used to demonstrate the practical robustness. Timing results on large random examples illustrate the potential speedup on current multicore machines.

1 Introduction

We are interested in the efficient and stable factorization of large sparse symmetric indefinite matrices. Most algorithms for this employ supernodes (see, for example, [1, 11, 12, 13, 18, 20]). That is, a set of consecutive columns having the same (or similar) sparsity pattern in the factor. By storing only those rows that contain nonzeros, each supernode may be held as a dense $n \times p$ trapezoidal matrix A of the form

$$(1.1)$$

This matrix is termed a *supernodal* matrix. In general, at non-root nodes, $n \gg p$.

The major numerical tasks to be performed on each supernode are:

Factor $A_{11} = L_{11}D_{11}L_{11}^T$;

Solve $L_{21} = A_{21}(D_{11}L_{11}^T)^{-1}$;

Form $S = L_{21}D_{11}L_{21}^T$ (Schur complement); and

Scatter S across other supernodes (using either multifrontal or supernodal techniques).

Here L_{11} is unit lower triangular, and D_{11} is block diagonal with 1×1 and 2×2 blocks. In practice, permutations are used for pivoting, however we omit these above for clarity of notation. For numerical stability, when selecting pivots the factor task needs to take account of the values of the entries in A_{21} as well as those in A_{11}. For this reason, the factor and solve tasks are often combined into a single kernel.

In addition to the scatter task, the key difference from an otherwise equivalent dense factorization is that pivots are only selected from within A_{11}. If a candidate pivot is found to be unsuitable, it is moved to a later supernode for elimination, with a guarantee that all pivots will be eliminated in the final supernode. Such pivots are said to be *delayed*. They generate additional floating-point operations and storage requirements. If pivots were instead chosen from A_{21}, much larger amounts of additional storage and computation would be required.

With the advent of manycore processors and the growing gap between the speed of communication and computation, many algorithms must be rewritten to reflect the changing balance in resource. As the pivoting decisions must be taken in a serial fashion, they are highly sensitive to the latency and speed of any communication or bandwidth costs incurred. With current algorithms that take account of the entries in A_{11} and A_{21}, all threads working on a supernode must endure stop-start parallelism for every column of the supernode. Even when running serially, performance issues are encountered if the entire supernode does not fit in the smallest level of cache.

This paper seeks to address these issues by developing effective pivoting strategies that significantly reduce the amount of communication required. A provably stable algorithm and a heuristic algorithm are presented; we will refer to these algorithms as *compressed threshold pivoting* algorithms. The heuristic algorithm is faster than the provably stable alternative and it more accurately approximates the behaviour of traditional threshold partial pivoting in terms of modifications to the pivot sequence. While it can demonstrably fail to control the growth factor for some pathological constructed examples, in practice it achieves numerical robustness even on the most difficult practical problems.

The rest of this paper proceeds as follows. In Section 2, the standard threshold partial pivoting technique used in many current sparse symmetric indefinite codes is reviewed. Section 3 explores the applicability of recent work on communication-reducing pivoting for dense factorizations to the sparse case and reviews techniques that are currently used for sparse problems. The new compressed threshold pivoting algorithms are introduced in Section 4; stability and communication are analysed in Sections 5 and 6, respectively. Numerical experiments are presented in Section 7 and conclusions are summarized in Section 8.

2 Threshold partial pivoting (TPP) within a sparse direct solver

In this section, we recall how threshold partially pivoting (TPP) may be incorporated within the combined factor and solve tasks.

The algorithm (which we refer to as the *threshold pivoting algorithm*) is applied to the supernodal matrix of (1.1) and tries to select up to p pivots from the first p rows (A_{11}). The entries in the remaining rows are used when testing for stability. As pivots can only be selected from A_{11}, traditional partial pivoting is not applicable. Instead, a threshold test is employed to limit the growth of entries in the factors. For a 1×1 pivot, the test on the suitability of column q is

$$|a(q,q)| \geq u \max_{i>q} |a(i,q)|, \tag{2.1}$$

Listing 1: Threshold partial pivoting (TPP) algorithm

Input: $a(1{:}n, 1{:}p)$ with $n \geq p$; parameters u and $small$
$nelim = 0$ // Number of eliminated variables
$m = 2$ // Index of current pivot candidate column
do while (elimination still possible)
   if ($\max(|a(m{:}n, m)|) < small$) **then**
      permute m to position $nelim + 1$
      record a zero pivot; $nelim = nelim + 1$// Special 1×1 case
      $m = m + 1$; **cycle** // Move to next column
   end if
   Find column index t of largest entry in $|a(m, nelim{+}1{:}m{-}1)|$

   // Try (t,m) as a 2×2 pivot
   $maxt = \max\{|a(i,t)| : i \geq nelim + 1, i \neq t, m\}$
   $maxm = \max\{|a(i,m)| : i \geq nelim + 1, i \neq t, m\}$
   if (test_2x2 $(a, m, t, maxm, maxt)$) **then**
      permute t and m to positions $nelim + 1$ and $nelim + 2$
      perform 2×2 pivot
      $nelim = nelim + 2$
      update $a(nelim{+}1{:}n, nelim{+}1{:}p)$
      $m = m + 2$; **cycle** // Move to next column
   end if

   // Failed as 2×2 pivot, try as 1×1
   $maxm = \max\{|a(i,m)| : i \geq nelim + 1, i \neq m\}$
   if ($|a(m,m)| \geq u * maxm$) **then**
      permute m to position $nelim + 1$
      perform 1×1 pivot
      $nelim = nelim + 1$
      update $a(nelim{+}1{:}n, nelim{+}1{:}p)$
      $m = m + 1$; **cycle** // Move to next column
   end if
end do

// Return true if (t,m) is a good 2×2 pivot, false otherwise
function test_2x2 $(a, t, m, maxm, maxt)$
   if ($\max(|a(t,t)|, |a(t,m)|, |a(t,t)|) < small$) **then return** false

   // Next test ensures 2×2 candidate is not singular and cancellation
   // does not adversely affect the calculation of its inverse
   $detscale = 1/\max(|a(t,t)|, |a(t,m)|, |a(t,t)|)$
   $detpiv1 = (a(t,m) * detscale) * a(t,m)$
   $detpiv0 = a(m,m) * detscale * a(t,t)$
   $detpiv = detpiv0 - detpiv1$
   if ($|detpiv| > \max(small, |detpiv0|/2, |detpiv1|/2)$) **then return** false

   if ($\max(maxm, maxt) < small$) **return** true
   if ($detpiv^{-1} * \left(detscale * \left| \begin{pmatrix} a(m,m) & a(t,m) \\ a(t,m) & a(t,t) \end{pmatrix} \right| \right) \begin{pmatrix} maxm \\ maxt \end{pmatrix} \leq u^{-1}$) **return** true
end function test_2x2

where $a(i, j)$ are the entries of the supernodal matrix and we are assuming that columns $1, 2, ...q - 1$ have already been pivoted on (see, for example, [7]). Similarly, for a 2×2 pivot, the test on the suitability of columns q and $q + 1$ is

$$\left| \begin{pmatrix} a(q,q) & a(q,q+1) \\ a(q,q+1) & a(q+1,q+1) \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{i>q+1} |a(i,q)| \\ \max_{i>q+1} |a(i,q+1)| \end{pmatrix} \leq \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}, \quad (2.2)$$

where the absolute value notation for a matrix refers to the matrix of corresponding absolute values (see, for example, [19]). The choice of the threshold parameter u ($0 < u \leq 0.5$) controls the balance between stability and sparsity in the factors, with a small u leading to a potentially large growth factor but preserving sparsity. Observe that both tests require a scan of the candidate column(s), which must be up-to-date (all operations from previous pivots must have been applied). Stability of the factorization of symmetric indefinite systems was considered by Ashcraft, Grimes and Lewis [2], who showed that bounding the size of the entries of L, together with a backward stable scheme for solving 2×2 linear systems, suffices to show backward stability for the entire process. Note that they found the widely used strategy of Bunch and Kaufmann [5] does not have this property.

Listing 1 outlines the kernel for performing the combined factor and solve tasks that is used within our recent multifrontal solver HSL_MA97 [13, 15]. The description uses the notation $a(i:j, r:s)$ to denote the submatrix consisting of rows i to j and columns r to s. In addition to the threshold pivoting parameter u (default value 0.01), the user-specified parameter $small$ (default 10^{-20}) is provided. All entries less than $small$ are treated as zero for the purposes of pivot selection. It is worth noting the care required to invert the 2×2 pivot in a stable fashion, whereby the pivot is scaled such that the largest entry is unity and the test $|detpiv| > \max(small, |detpiv0|/2, |detpiv1|/2)$ ensures both that the pivot is nonsingular and that cancellation does not occur.

The TPP algorithm is similar to that used by, for example, other HSL [15] sparse symmetric indefinite solvers but has a preference for 2×2 pivots over 1×1 pivots (see also [19]). While this description is written in a right-looking fashion (i.e. the unelimated part of the matrix is updated after each pivot selection), the actual implementation uses multiple levels of blocking, some of which use left-looking updates rather than right-looking ones for performance reasons (details are given in [13]). However, a key feature is that, at any given time, columns $nelim + 1$ to m must all be up-to-date, where $nelim$ is the number of pivots selected so far and m is the index of the current candidate pivot column. Candidate pivots are only permuted to the front of the matrix once they have been accepted.

2.1   Parallel variants

In this paper, we will compare our new algorithms against the following three parallel variants of Listing 1 with different communication patterns. We assume that P processors are used.

TPP outer update (TPP_OU) The supernodal matrix (1.1) is divided into block columns, each of width nbi (with the width of the last block column adjusted as necessary). The supernodal factorization proceeds serially using left-looking updates within each block column. Upon completion of the factorization of a block column, a parallel right-looking update of the remaining block columns is performed.

TPP Variant A The n rows of the supernodal matrix A (including A_{11}) are split equally between the processors. The processor that owns the current pivot row m determines its pairing t and communicates the submatrix $\begin{pmatrix} a(t,t) & a(t,m) \\ a(t,m) & a(m,m) \end{pmatrix}$ to each of the other processors. Each processor k finds local maximum values $maxm_k$ and $maxt_k$ that are then reduced in parallel to find a global $maxm$ and $maxt$. The acceptance test and pivoting operations are performed locally, and the owner of the pivotal rows broadcasts the part of A_{11} needed for the update to all other processors. The local updates are then performed.

TPP Variant B The $n - p$ rows of A_{21} are split equally between the processors. The p rows of A_{11} are replicated on every processor. Each processor independently finds the same pivots m and t and calculates local maximum values $maxm_k$ and $maxt_k$ that are then reduced in parallel to find a global $maxm$ and $maxt$. The acceptance test, pivoting operation and updates of all locally stored rows are then performed without need for further communication.

We consider parallel schemes in which each processor controls $O(p^2)$ data and performs $O(p^3)$ operations. As such, supernodes with $n \gg p$ are of particular interest. In this case, each of the above variants performs at least one communication per pivot, which can be hidden by at most $O(p^2)$ operations, or less if blocking is used to exploit the cache architecture of individual processors. To improve on this, we need to consider ways of performing more pivot operations per communication.

3 Existing methods

In this section, we briefly review techniques that have been proposed to overcome the problem of pivoting in parallel, both in the dense case and in the sparse case. We consider their suitability for sparse indefinite systems.

3.1 Dense

In the dense case, pivots may be chosen from within A_{21} as well as A_{11}. A number of different pivoting schemes have been proposed, including pairwise pivoting [3, 22] and parallel pivoting [23], and block variants thereof. Parallel pivoting is unstable. While pairwise pivoting is more stable, the growth factor is more than linear with respect to the matrix size [10]. Sparse versions of these algorithms are possible but have not been studied as they are likely to suffer from the same problems as the dense variants.

A more radical alternative pivoting technique has recently been introduced by Grigori, Demmel and Xiang [10]. Their CALU algorithm uses *tournament pivoting* whereby the supernode is recursively bisected into sections upon which an LU factorization is performed to select the best p pivot rows. This is demonstrated in Figure 3.1 and motivates our approach in this paper. For each block, an LU factorization is performed to identify the best p pivot rows within that block. These rows are then concatenated with those selected from a partner block and the process repeated. Once the full reduction tree has been evaluated, the selected pivots are used for the factorization of the supernode. While this technique provides weaker guarantees upon growth than traditional partial pivoting, it is no worse than partial pivoting applied at a block level. An analysis is presented in [10] showing that this algorithm performs an optimal amount of communication that is asymptotically less than Gaussian elimination with partial pivoting. Hence, it is faster on platforms where communication is expensive. Furthermore, the method has been shown to be stable in practice. However, as pivots are selected from within A_{21}, it is not applicable to the sparse case.

Other approaches seek to avoid the need for pivoting altogether. Becker, Baboulin and Dongarra [4] use a randomizing scaling with structure similar to a fast-Fourier transform to homogenise the matrix such that the values are sufficiently uniformly distributed that pivoting is almost surely not required. This technique of using *recursive butterfly matrices* cannot be applied to the sparse case as they lead to complete fill-in of the scaled matrix SAS.

3.2 Sparse

Since searching candidate columns is expensive, an obvious remedy is to restrict the search and possibly risk sacrificing some stability. In a *restricted pivoting* approach, the check for large entries is restricted to examining only the values of the entries in A_{11}, completely ignoring those in A_{21}. As it may not be possible to find a suitable pivot from within the diagonal block, the sparse direct solver PARDISO [20] uses restricted pivoting with a pivot perturbation strategy (called *static pivoting*), which is similar to that employed in the unsymmetric case in [17]. Static pivoting allows no pivots to be delayed, thereby greatly simplifying

Figure 3.1: Tournament pivoting on a reduction tree as used by CALU.
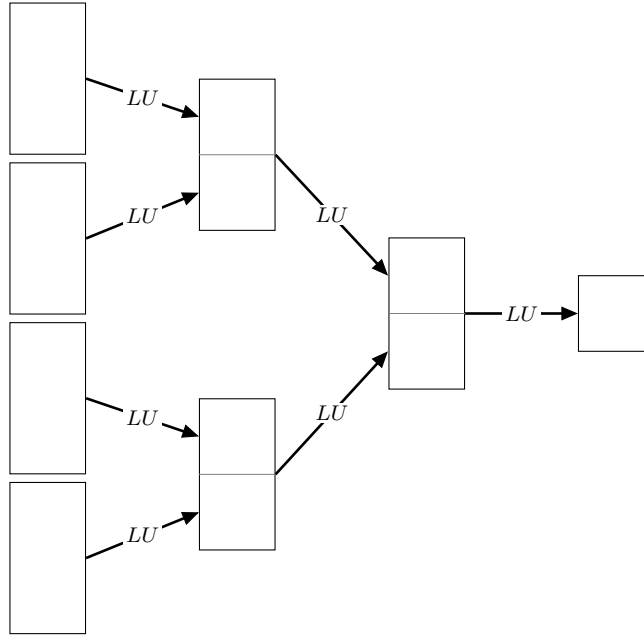
the coding compared with a direct solver that does permit delayed pivots, as well as limiting the fill in the factors and the operations required to compute them. However, since the factorization may not be accurate, it is often necessary to perform a number of steps of refinement to try and recovery accuracy but this is not guaranteed to be successful. However, as discussed by Schenk, Wächter and Hagemann in [21], reliability may be improved by using a matching-based ordering. This aims to bring large entries close to the diagonal, with the hope that they will make suitable block pivot candidates (see also Duff and Pralet [9]). While this extends the class of problems that can be solved, there are some particularly tough indefinite linear systems that we have been unable to solve successfully using this approach (see [14]). Furthermore, some applications, particularly those arising from optimization, require an accurate measure of the matrix inertia that can be difficult to obtain when static pivoting is used.

Kim and Eijkhout [16] present a *try-it-and-see* approach to pivoting in their code for hp-adaptive finite element problems. Restricted pivoting is used at each node of the assembly tree, but an a posteriori check is made for growth below the diagonal block. Should excessive growth be detected, the supernodal matrix is reconstructed from the contribution blocks of its child nodes and pivots are delayed to its parent node.

4 Compressed threshold pivoting

As stated at the end of Section 2, we need to reduce the number of communications per pivot. For $n \gg p$, we propose the construction of a small representative matrix that can be used to make pivoting decisions without the need to work with the full supernodal matrix (and hence avoids the need to communicate with other processors). Henceforth we shall refer to this representative matrix as the *compressed matrix C*. For the purposes of this paper, we shall consider only the case where C is $p \times p$, but there is no requirement that this is so. The application of (potentially modified) threshold partial pivoting to the small trapezoidal matrix comprising the diagonal block A_{11} of (1.1) and the matrix C as below is considered.

$$(4.1)$$

The factorization of (4.1) establishes a permutation and factors L_{11} and D_{11} that can be applied to A_{21} (in parallel) without the need to perform any further pivoting. This process is summarized in Figure 4.1.

We present two methods of constructing (and updating) the compressed matrix. The *strict* method is numerically stable, however it can be too pessimistic (that is, lead to a large number of delayed pivots that TPP would have selected) so we also introduce the *relaxed* method that can be unstable for some matrices but, in practice, if used with an appropriate scaling and ordering, will be shown to be stable (see Section 7.2).

Both methods construct their compressed matrix in parallel using a tree reduction similar to that previously described for the CALU algorithm. The LU reduction operation of Figure 3.1 is replaced by the techniques that we describe in the next two sections.

4.1   Strict compressed pivoting

In the strict method, we first partition the rows of A_{21} into sets corresponding to the column in which their entry of largest absolute value lies. Thus all the indices of the rows in A_{21} that have their maximum entry in column j belong to the set J_j given by

$$J_j = \{i : j = \arg\max_k |a(i, k)|, i > p\}.$$

Ties are resolved in favour of the lowest value of k. The compressed matrix $C = \{c(j, k)\}$ is constructed row-by-row from these sets:

$$c(j, k) = \begin{cases} \max_{i \in J_j} |a(i, k)|, & J_j \neq \phi, \\ 0, & \text{otherwise.} \end{cases}$$
$$(4.2)$$

That is, row j of C is determined by the columnwise maxima of J_j. This is illustrated in Figure 4.2.

The factorization proceeds as per threshold partial pivoting applied to (4.1). However, the steps involving the application of pivots and the updating of the trailing submatrix are modified for C (but not for A_{11}). For threshold partial pivoting we would use the following update formulae for a 1×1 pivot:

$$\begin{aligned} \hat{c}(:, k) &= c(:, k)/a(k, k) \\ \hat{c}(:, k{+}1{:}p) &= c(:, k{+}1{:}p) - c(:, k)a(k, k)a(k{+}1{:}p, k)^T. \end{aligned}$$
$$(4.3)$$

We want row j of C to represent the worst possible growth in the rows J_j. Using similar ideas to those involved in the 2×2 pivot test (2.2), we modify (4.3) to use absolute values throughout (exploiting the fact that all elements of C are positive):

$$\begin{aligned} \hat{c}(:, k) &= c(:, k)/|a(k, k)| \\ \hat{c}(:, k{+}1{:}p) &= c(:, k{+}1{:}p) + c(:, k)\,|a(k, k)|\,|a(k{+}1{:}p, k)|^T. \end{aligned}$$
$$(4.4)$$

A proof of backwards stability is given in Section 5.

We observe that we experimented with using a single row to represent the entire A_{21} matrix (based on the set $J = \{1, 2, \ldots, n - p\}$, that is, C is constructed by taking the entry of largest absolute value in each row of A_{21}), but found this led to the rejection of almost all pivots beyond the first few. By using multiple rows, the over estimation of the growth can be controlled.

7

Figure 4.1: Compressed threshold pivoting

(a)
Analysis of A_{21}

(b)
Construction of
compressed matrix C

(c)
Factor
$\begin{pmatrix} A_{11} \\ C \end{pmatrix} = (P\hat{L})D(P\hat{L})^T$

(d)
Apply permutation
P to A_{21}

(e)
Solve
$D_{11}L_{11}^T L_{21} = PA_{21}$

A_{11}

Factor

A_{11}
C

Factor

P
L_{11}
$\hat{C}$
D_{11}

P
D_{11}

PA_{21}
L_{11}
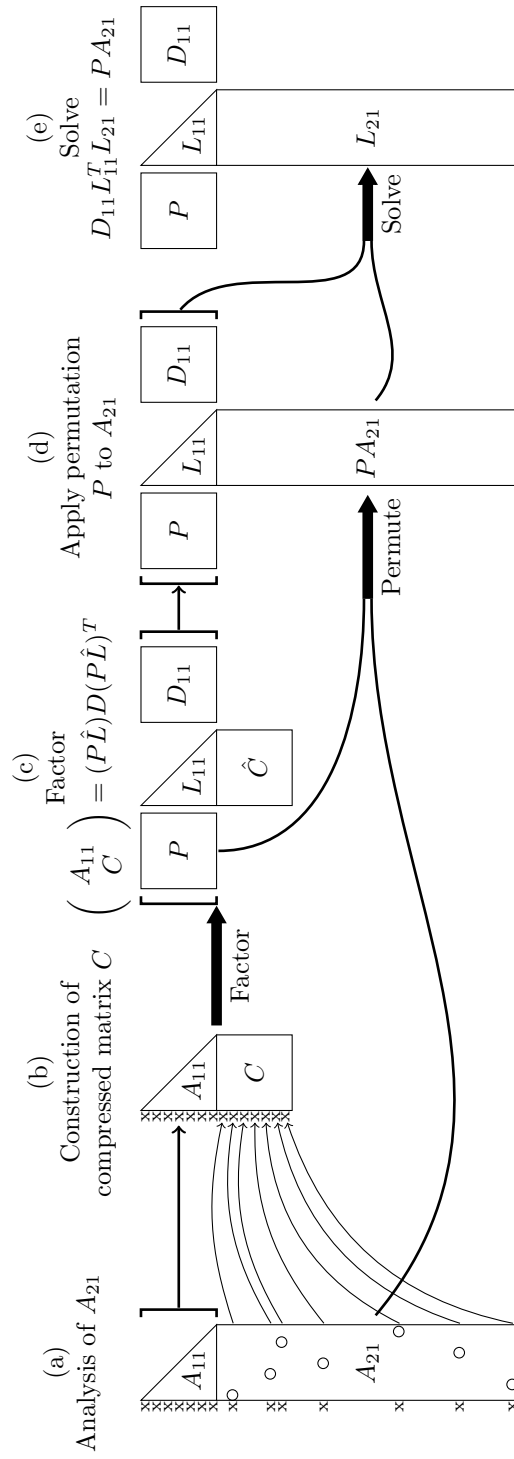
Permute

P
L_{11}
D_{11}

L_{21}

Solve

A_{21}

Figure 4.2: Example of strict compressed matrix construction. Observe that $J_1 = \phi$. Bold is used to indicate row maxima of A_{21} and column maxima of A_{J_2} and A_{J_3}.

$$
A_{21} = \begin{pmatrix} 1 & \mathbf{10} & 10 \\ 2 & 3 & 4 \\ & 10 & -3 \\ 4 & -\mathbf{5} & 4 \\ & -6 & \mathbf{8} \end{pmatrix}
\qquad
A_{J_2} = \begin{pmatrix} 1 & \mathbf{10} & \mathbf{10} \\ & 10 & -3 \\ \mathbf{4} & -5 & 4 \end{pmatrix}
\qquad
C = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 10 & 10 \\ 2 & 6 & 8 \end{pmatrix}
$$

$$
A_{J_3} = \begin{pmatrix} \mathbf{2} & 3 & 4 \\ & -\mathbf{6} & \mathbf{8} \end{pmatrix}
$$

4.2 Relaxed compressed pivoting

For the relaxed method, we make the experimental observation that the column maxima often remain in the same position as the factorization progresses. Even in those cases where the locations of the maxima vary, the value at the old location is often close to that at the new.

For each column of the matrix (1.1), we include in the compressed matrix a row that contains the entry of largest absolute value in that column at the start of the supernode factorization. The hope is that this row carries sufficient information to reject unstable pivots using the standard tests and normal update formulae (as given in (4.3)). However, the method risks using unstable pivots rather than rejecting acceptable ones.

The algorithm for constructing C is as follows. First mark all rows of A_{21} as unflagged. Then, for each column j $(1 \leq j \leq p)$, find the the entry of largest magnitude in A_{21} outwith an already flagged row. Flag the corresponding row, include it in C, and continue to the next column. The results of this algorithm are demonstrated in Figure 4.3.

Figure 4.3: Example of relaxed compressed matrix construction. Bold is used to indicate the entries of A_{21} used in the selection of rows for inclusion in C.

$$
A_{21} = \begin{pmatrix} 1 & \mathbf{10} & 10 \\ 2 & 3 & 4 \\ & 10 & -3 \\ \mathbf{4} & -5 & 4 \\ & -6 & \mathbf{8} \end{pmatrix}
\qquad
C = \begin{pmatrix} 4 & -5 & 4 \\ 1 & 10 & 10 \\ & -6 & 8 \end{pmatrix}
$$

Note that by insisting on flagging a different row for each column, p rows are always marked, and ordering affects tie-breaking (for example, in Figure 4.3 encountering row 3 before row 1 would have resulted in a different C). The choice to add the complication of flagging is based on practical experience. Otherwise, if for several columns the entry of largest absolute value occurred in the same row, the total number of included rows could be significantly fewer than p. Experiments showed that in this case the resulting factorization was less stable for a number of problems tested.

5 Stability analysis

Following the stability analysis presented by Ashcraft, Grimes and Lewis [2], we define the partially factorized supernodal matrix $A^{(q)}$ as that formed after q eliminations and their updates have been applied to (1.1), with $A^{(0)} = A$. Let

$$
\mu_q = \max_{i,j} |a^{(q)}(i,j)|
$$

be the maximum absolute value of an entry of $A^{(q)}$. We proceed to analyse the application of the algorithm given in Listing 1 to (4.1). The analysis assumes that the pivotal columns t and m have been permuted to

positions q and $q + 1$, that is to be the first columns in the unelimineted part of the matrix. We seek to demonstrate that the entries of the factor L are bounded, and that growth in $A^{(q)}$ (and hence D) is limited:

$$\mu_{q+1} < \mu_q \left(1 + u^{-1}\right). \tag{5.1}$$

We first note that (5.1) does not hold for relaxed compressed pivoting. To illustrate this, consider the following factorization, where ϵ is small,

$$A = \begin{pmatrix} 1 & -1 & & \\ -1 & 2 & & \\ u^{-1} & & & \\ & & u^{-1} & \\ \hline u^{-1} - \epsilon & u^{-1} - \epsilon & & \end{pmatrix} \quad \Rightarrow \quad L = \begin{pmatrix} 1 & & & \\ -1 & & 1 & \\ u^{-1} & & u^{-1} & \\ & & u^{-1} & \\ \hline u^{-1} - \epsilon & & 2(u^{-1} - \epsilon) & \end{pmatrix}.$$

The entries below the line are not included in the compressed matrix, so are not tested for stability. Observe that after p steps, we can have an entry of L that is close to pu^{-1}. This means that L can be effectively unbounded.

We next analyse strict compressed pivoting. We have the following bound on the entries of $A^{(q)}_{21}$.

Lemma 5.1. *Let the compressed matrix C be defined by (4.2) and let $C^{(q)}$ to be the matrix C after q eliminations ($C^{(0)} = C$). Then for $q \geq 0$,*

$$|a^{(q)}(i,k)| \leq c^{(q)}(j,k) \quad \text{for all} \quad i \in J_j, 1 \leq j, k \leq p.$$

Proof: This is by construction for $q = 0$, and induction on the update equations (4.4) for $q > 0$ (for notational convenience here and elsewhere we drop the superscript on the reduced matrix at step q, but not for other steps):

$$a^{(q+1)}(i,k) = a(i,k) - \frac{a(i,q)a(k,q)}{a(q,q)}$$

$$\Rightarrow \left| a^{(q+1)}(i,k) \right| \leq c(j,k) + \frac{c(j,q)|a(k,q)|}{|a(q,q)|} = c^{(q+1)}(j,k)$$

$\square$

This lemma is used to prove strict compressed pivoting is backwards stable.

Theorem 5.1. *For strict compresssed pivoting the bound (5.1) holds and the entries of L are bounded above by u^{-1}.*

Proof: We proceed as in [2] and consider 1×1 and 2×2 pivots separately.

1×1 pivots. Define

$$\gamma_q = \max \left(\max_{q < i \leq p} |a(i,q)|, \max_{1 \leq i \leq p} c(i,q) \right).$$

Then if $a(q,q)$ is a 1×1 pivot it satisfies $a(q,q) \geq u\gamma_q$. The entries of $A^{(q)}$ are given by

$$a^{(q+1)}(i,k) = a(i,k) - \frac{a(i,q)a(k,q)}{a(q,q)}.$$

10

Using Lemma 5.1, these may be bounded by

$$
|a^{(q+1)}(i,k)| \quad \leq \quad
\begin{cases}
|a(i,k)| + \dfrac{|a(i,q)|\,|a(k,q)|}{|a(q,q)|} & i \leq p \\[3mm]
|a(i,k)| + \dfrac{|c(i,q)|\,|a(k,q)|}{|a(q,q)|} & i > p
\end{cases}
$$

$$
\leq \quad |a(i,k)| + \frac{\gamma_q^2}{|a(q,q)|}.
$$

Taking maximums over i,k yields

$$
\mu_{q+1} \quad \leq \quad \mu_q + \frac{\gamma_q^2}{|a(q,q)|}
$$

$$
\leq \quad \mu_q + \gamma_q u^{-1} \leq \mu_q \left(1 + u^{-1} \right),
$$

and $A^{(q+1)}$ has its growth bounded. Similarly, the entries of L are bounded,

$$
|l(i,q)| \quad = \quad \frac{|a(i,q)|}{|a(q,q)|} \leq u^{-1} \frac{a(i,q)}{\gamma_q} \leq u^{-1}.
$$

2×2 pivots. Define

$$
\gamma_q \quad = \quad \max \left(\max_{q+1 < i \leq p} a(i,q),\ \max_{1 \leq i \leq p} c(i,q) \right),
$$

$$
\gamma_{q+1} \quad = \quad \max \left(\max_{q+1 < i \leq p} a(i,q+1),\ \max_{1 \leq i \leq p} c(i,q+1) \right),
$$

$$
D_q \quad = \quad \begin{pmatrix} a(q,q) & a(q,q+1) \\ a(q,q+1) & a(q+1,q+1) \end{pmatrix}.
$$

Then the nonsingular pivot D_q satisfies

$$
|D_q^{-1}| \begin{pmatrix} \gamma_q \\ \gamma_{q+1} \end{pmatrix} \leq \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}.
$$

The entries of $A^{(q+2)}$ are given by

$$
a^{(q+2)}(i,k) \quad = \quad a(i,k) + \begin{pmatrix} a(k,q) & a(k,q+1) \end{pmatrix} D_q^{-1} \begin{pmatrix} a(i,q) \\ a(i,q+1) \end{pmatrix}.
$$

Again, using Lemma 5.1, these may be bounded by

$$
|a^{(q+2)}(i,k)| \quad \leq \quad
\begin{cases}
|a(i,k)| + \begin{pmatrix} |a(k,q)| & |a(k,q+1)| \end{pmatrix} |D_q^{-1}| \begin{pmatrix} a(i,q) \\ a(i,q+1) \end{pmatrix} & i \leq p \\[3mm]
|a(i,k)| + \begin{pmatrix} |a(k,q)| & |a(k,q+1)| \end{pmatrix} |D_q^{-1}| \begin{pmatrix} c(i,q) \\ c(i,q+1) \end{pmatrix} & i > p
\end{cases}
$$

$$
\leq \quad |a(i,k)| + \begin{pmatrix} |a(k,q)| & |a(k,q+1)| \end{pmatrix} |D_q^{-1}| \begin{pmatrix} \gamma_q \\ \gamma_{q+1} \end{pmatrix}
$$

$$
\leq \quad |a(i,k)| + \begin{pmatrix} |a(k,q)| & |a(k,q+1)| \end{pmatrix} \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}.
$$

Taking maximums over i, k yields

$$\begin{aligned}
\mu_{q+2} &\leq \mu_q + u^{-1}\left(|a(k,q)| + |a(k,q+1)|\right) \\
&\leq \mu_q(1 + 2u^{-1}).
\end{aligned}$$

Likewise the entries of L are bounded,

$$\begin{aligned}
\left(\begin{array}{cc} l(i,q) & l(i,q+1) \end{array}\right) &= \left(\begin{array}{cc} a(i,q) & a(i,q+1) \end{array}\right) D_q^{-T} \\
\left(\begin{array}{cc} |l(i,q)| & |l(i,q+1)| \end{array}\right) &\leq \left(\begin{array}{cc} \gamma_q & \gamma_{q+1} \end{array}\right) \left|D_q^{-T}\right| \\
&\leq \left(\begin{array}{cc} u^{-1} & u^{-1} \end{array}\right).
\end{aligned}$$

$\square$

6 Communication analysis

Appendix A presents an analysis of communication costs using a model of a parallel machine. We count the total number of operations and total amount of bandwidth required. We also derive a count of the number of messages required as a count the number of communication latencies that are necessarily incurred if all other operations take zero time. We assume that each pivot is accepted as soon as it is encountered (and hence permutations are not applied).

Table 6.1 gives exact results for factorizing an $n \times p$ supernodal matrix on P processors. Under the assumption that $P = O(n)$, Table 6.2 summarises the results using order notation. The operation counts are given in terms of the number of additional operations above those required for traditional threshold partial pivoting,

$$\text{TPP}_{\text{ops}}(n, p) = \frac{29}{6}p - \frac{3}{4}p^2 - \frac{1}{3}p^3 + 2np + \frac{1}{2}np^2.$$

We observe that restricted pivoting provides a theoretical bound on the number and amount of communication for a parallel code of the type considered here, as it performs no communication for pivoting. The main gain from using compressed pivoting compared with Variants A and B is the factor p reduction in the number of messages sent, at the cost of using five times more bandwidth. In terms of operations, only Variant B significantly increases the (asymptotic) operation count above that of threshold partial pivoting, although strict compressed pivoting does add an extra Pp^2 term. However, these increased operation counts are somewhat misleading as they are spread across P processors.

We remark that, for a sparse direct solver, the costs must be summed over all the supernodes. If there are delayed pivots at a supernode then n and p will increase beyond that predicted by the analyse phase of the solver, leading to the need to perform permutations, the repeated testing of candidate pivots, additional operations and communication as well as denser factors. For strict compressed pivoting, the number of delayed pivots is usually greater than for relaxed compressed pivoting and can be prohibitive; this is illustrated by the results in Section 7.2.

Table 6.1: Parallel communication analysis results

	Operations	Messages	Bandwidth
TPP Variant A	$\text{TPP}_{\text{ops}}(n,p)$	$p + \frac{1}{2}p\log_2 P$	$-\frac{1}{2}p + \frac{1}{2}Pp(p+2)$
TPP Variant B	$\text{TPP}_{\text{ops}}(n,p) + (P-1)(\frac{29}{6}p + \frac{5}{4}p^2 + \frac{1}{6}p^3)$	$1 + \frac{1}{2}p\log_2 P$	$-\frac{1}{2}p(p+3) + \frac{1}{2}Pp(p+5)$
Strict Compressed	$\text{TPP}_{\text{ops}}(n,p) + \frac{1}{2}p((p-1)p+3) + n(2p-1) + Pp^2$	$1 + \log_2 P$	$-\frac{1}{2}p(5p+1) + \frac{1}{2}Pp(5p+1)$
Relaxed Compressed	$\text{TPP}_{\text{ops}}(n,p) + \frac{1}{2}p((p+2)p-2) + (n+P)p$	$1 + \log_2 P$	$-\frac{1}{2}p(5p+1) + \frac{1}{2}Pp(5p+1)$
Restricted	$\text{TPP}_{\text{ops}}(n,p) - p(n-p)$	1	$-\frac{1}{2}p(p+1) + \frac{1}{2}Pp(p+1)$

Table 6.2: Summary of parallel communication analysis results (assuming $P = O(n)$).

	Operations	Messages	Bandwidth
TPP Variant A	$O(np^2)$	$O(p \log n)$	$O(np^2)$
TPP Variant B	$O(np^3)$	$O(p \log n)$	$O(np^2)$
Relaxed Compressed	$O(np^2)$	$O(\log n)$	$O(np^2)$
Strict Compressed	$O(np^2)$	$O(\log n)$	$O(np^2)$
Restricted	$O(np^2)$	$O(1)$	$O(np^2)$

7　Numerical experiments

Because of the complexity involved in efficiently implementing all our algorithms and measuring timing performance within a real sparse solver, in Section 7.1 we present timing results for large random (dense) matrices that are constructed to avoid the need for pivoting. This allows us to simulate the performance overheads of various pivoting techniques in the best-case where no pivot candidates are rejected. However, we note that when a significant number of pivot candidates are rejected we would expect the advantage from using compressed pivoting techniques to increase as threshold partial pivoting would require additional communication to retest these pivots later in the factorization. To explore the reliability of the compressed and restricted pivoting algorithms, in Section 7.2 we present numerical stability results for a set of sparse problems arising from practical applications.

All experiments are performed on the machine summarised in Table 7.1.

Table 7.1: Description of machine used for numerical experiments

| **Processor**	$2 \times$ Intel Xeon E5-2687W
Physical Cores	16
Memory	64GB
Compiler	ifort 12.1.0
BLAS	MKL 10.3 update 6
L1/L2 cache (per core)	32KB / 256KB
L3 cache (shared)	20MB
Compiler flags	`ifort -O3 -xAVX -no-prec-div -ip`

7.1　Performance experiments on large random matrices

Figure 7.1 shows the performance characteristics for the threshold partial pivoting variants discussed in Section 2.1. For the TPP_OU variant we use block column size $nbi = 16$. The top graph shows a slice through the (n, p) parameter space for large fixed p, while the bottom graph shows a slice for large fixed n.

As we might expect, the figure shows that for small n the TPP_OU approach is the fastest because it avoids the communication overheads of the inner loops inherent in Variants A and B. For larger n, there is sufficient work to amortize such overheads and Variants A and B perform best, with a slight performance advantage for Variant B. As p increases, more time is spent in the outer update, and there is therefore little to choose between the parallel implementations.

Figure 7.2 compares the best (on a case-by-case basis) variant of threshold partial pivoting with the two proposed compressed schemes and restricted pivoting. Note that we cannot expect either of the compressed schemes to be faster than restricted pivoting because, as already noted, they always perform more operations and more communication. It is clear that as n and p increase, the compressed pivoting techniques substantially outperform the best threshold partial pivoting variant. For large n and p they are over twice

Figure 7.1: Performance of variants of threshold partial pivoting using 16 threads.
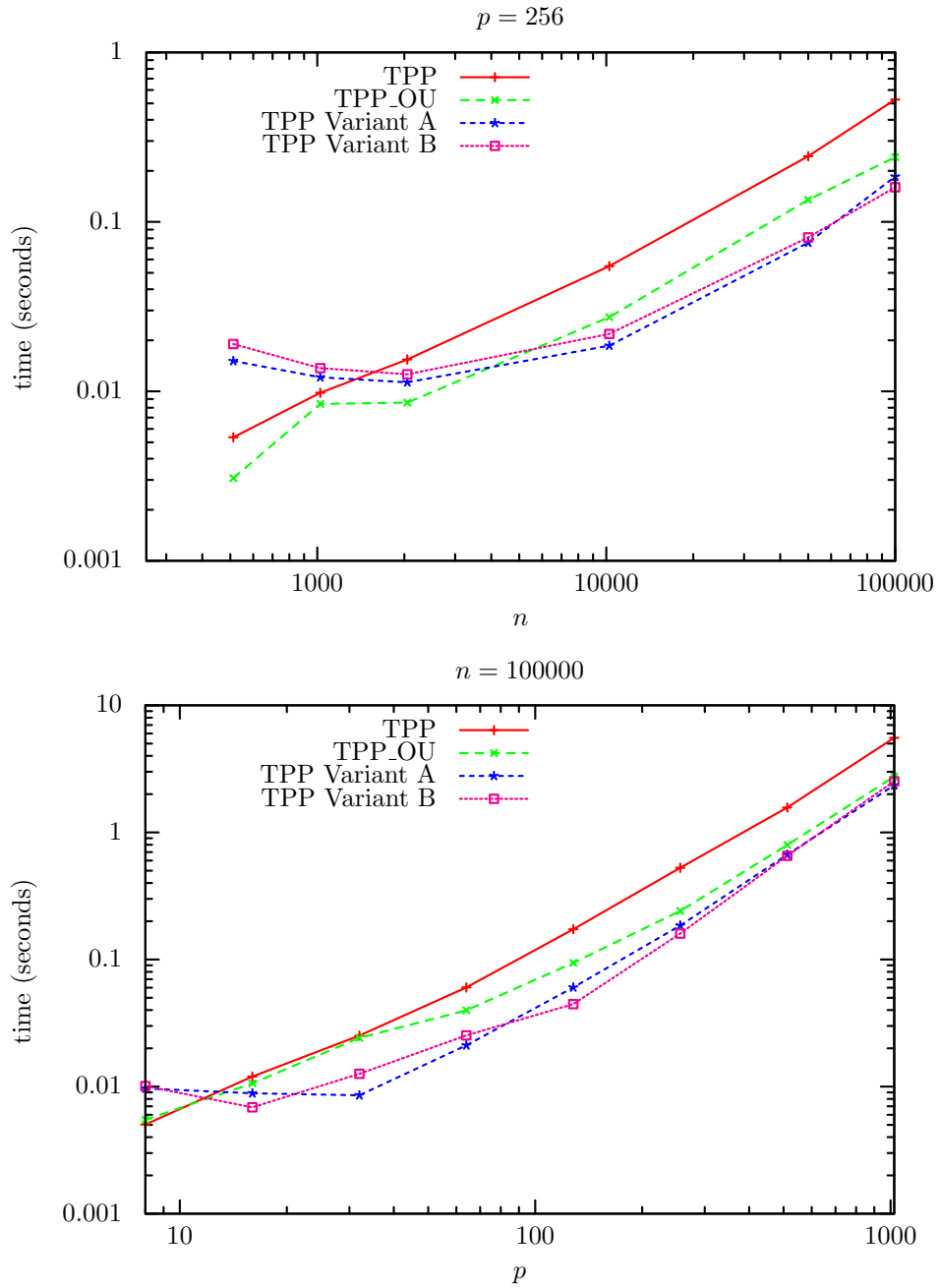
as fast. Further, they almost approach the performance of restricted pivoting while offering better numerical stability, as will be demonstrated in the next section.

7.2 Real-world numerical stability

We present results for two sets of 25 sparse indefinite problems drawn from the University of Florida Sparse Matrix Collection [6]. Test Set 1 consists of a selection of general problems of order at least 50000, while Test Set 2 consists of problems where threshold partial pivoting leads to a significant number of delayed pivots (these problems are selected from those surveyed in our recent study [14] on tough indefinite systems). The problems are scaled using a weighted matching approach (as implemented by MC64 [8]) and solved with a modified version of our sparse direct solver HSL_MA97 [13]. In all the tests, we use the default threshold parameter $u = 0.01$. By default, HSL_MA97 chooses between using an approximate minimum degree ordering and a nested dissection ordering (the choice is made on the basis of the order of the matrix and its density). However, it also offers a matching-based ordering. For tough indefinite problems, matching-based orderings can substantially reduce the number of delayed pivots albeit at the possible cost of additional operations and denser factors (see [14, 21]).

For the solution of the indefinite system $\mathcal{A}x = b$, Figure 7.3 plots the scaled backward error

$$\text{bwd err} = \frac{\|\mathcal{A}x - b\|_\infty}{\|\mathcal{A}\|_\infty \|x\|_\infty + \|b\|_\infty},$$

after 10 steps of iterative refinement for the pivoting strategies described in this paper (used with the default ordering). As expected, they each perform adequately on the general problems of Test Set 1. However, on the more numerically challenging problems of Test Set 2, the restricted pivoting approach fails (that is, iterative refinement fails to converge to a backward error smaller than 10^{-14}), while the two numerically stable approaches (threshold partial pivoting and strict compressed pivoting) solve every problem to machine precision. The relaxed compressed pivoting fails on 3 problems.

By applying a matching-based ordering to the problems in Test Set 2, we obtain the results presented in Figure 7.4. Restricted pivoting still fails to converge to an accurate answer for 13 of the 25 problems, but all failures for the relaxed compressed pivoting are eliminated. These results demonstrate that, although not backward stable, if combined with a matching-based ordering and scaling, relaxed compressed pivoting is stable in practice.

Figures 7.5 and 7.6 present the numbers of delayed pivots for the default and matching-based orderings, respectively. Strict compressed pivoting generally results in more delayed pivots than TPP, while for relaxed compressed pivoting the number is the same or fewer than for TPP. We remark that a small number of delayed pivots (typically less than 1000 for problems of the size used in our tests) has no significant effect on performance. The problems in Test Set 1 with the default ordering demonstrate the weakness of strict compressed pivoting. With the exception of problems GHS_indef/c-72 and GHS_indef/bmw3_2, threshold partial pivoting and relaxed compressed pivoting give few (if any) delayed pviots. But the stricter pivot selection of strict compressed pivoting results in the generation of over 1000 times more delayed pivots for some problems. In performance terms, for 5 of the 25 problems in Test Set 1 the HSL_MA97 time using strict compressed pivoting is more than twice that of using threshold partial pivoting (sometimes more than four times greater). Of course, for numerically straightforward problems such as these, the number of delayed pivots can be reduced by using a smaller threshold parameter u without compromising stability and this is an option that may want to be considered with strict compressed pivoting. Using a matching-based ordering also substantially reduces the number of delayed pivots (but may involve more operations and greater fill-in).

15

Figure 7.2: Performance of parallel pivoting schemes using 16 threads. For TPP the best variant is used for each combination of n and p.
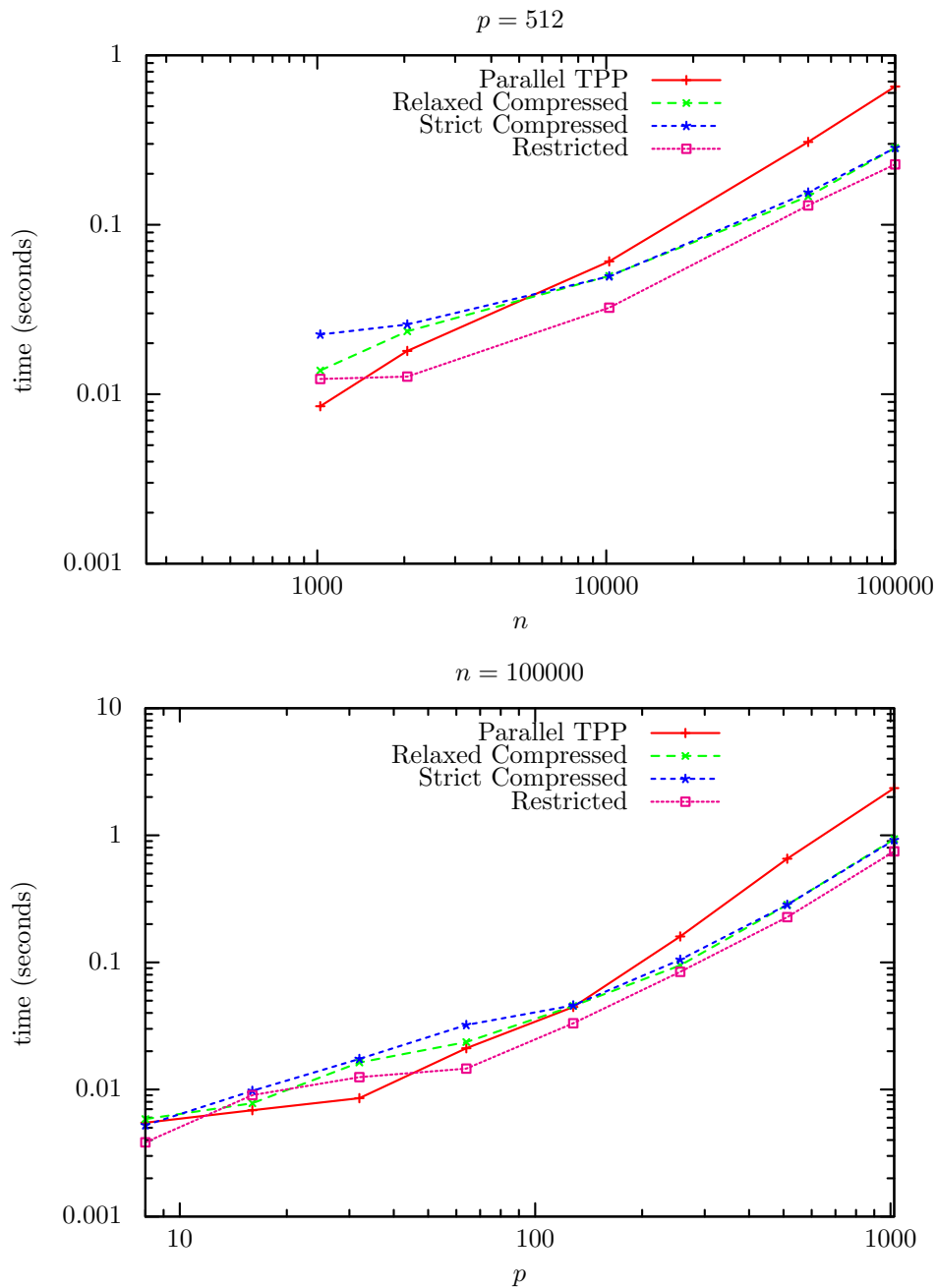

$p = 512$

$n = 100000$

Figure 7.3: Backward errors after iterative refinement (default ordering).
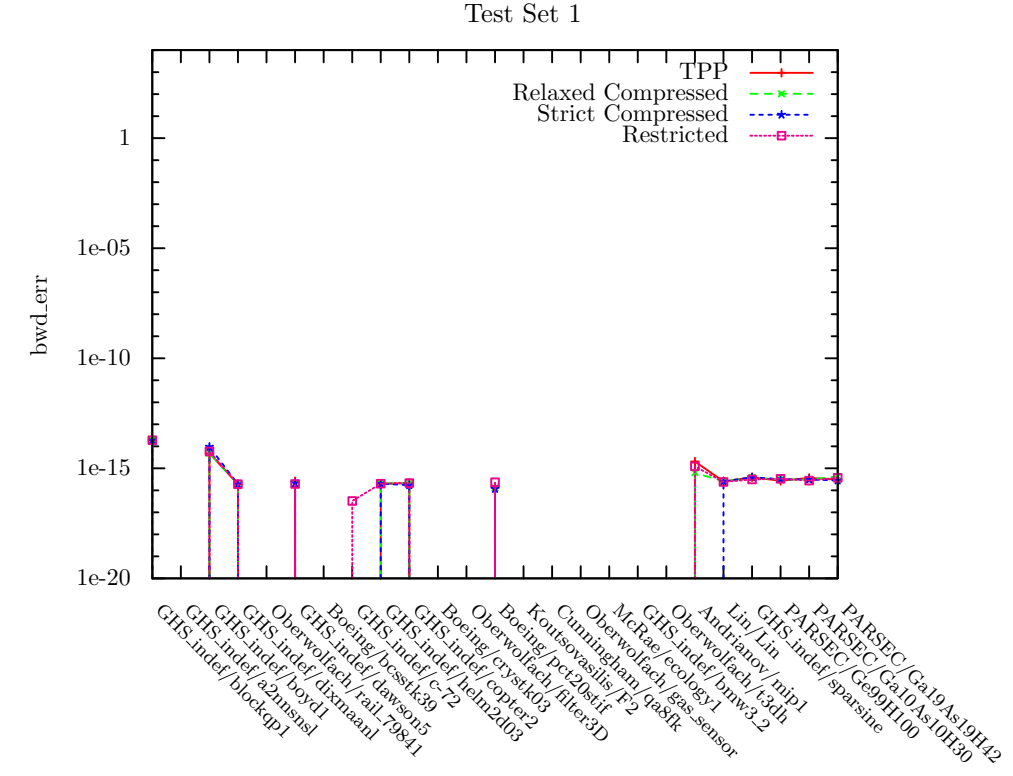Test Set 1

Test Set 2

Figure 7.4: Backward errors after iterative refinement (matching-based ordering).
Test Set 2

Figure 7.5: The number of additional delays generated by compressed pivoting compared with the number generated by threshold partial pivoting (default ordering).

Test Set 1

TPP   Relaxed   Strict
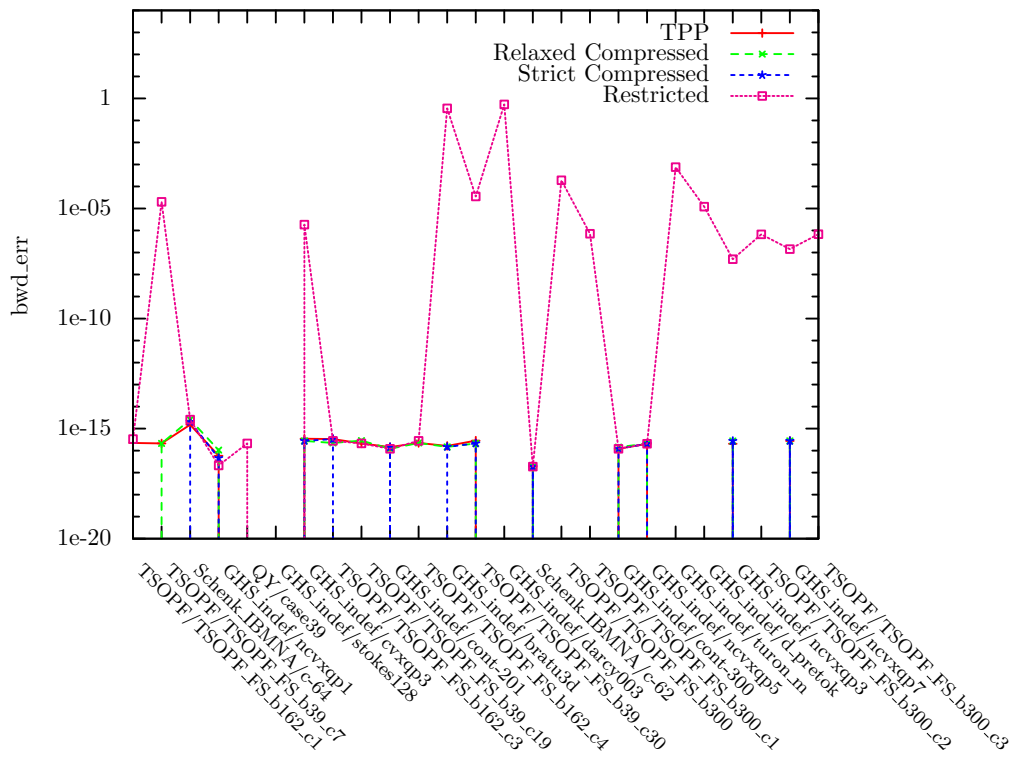

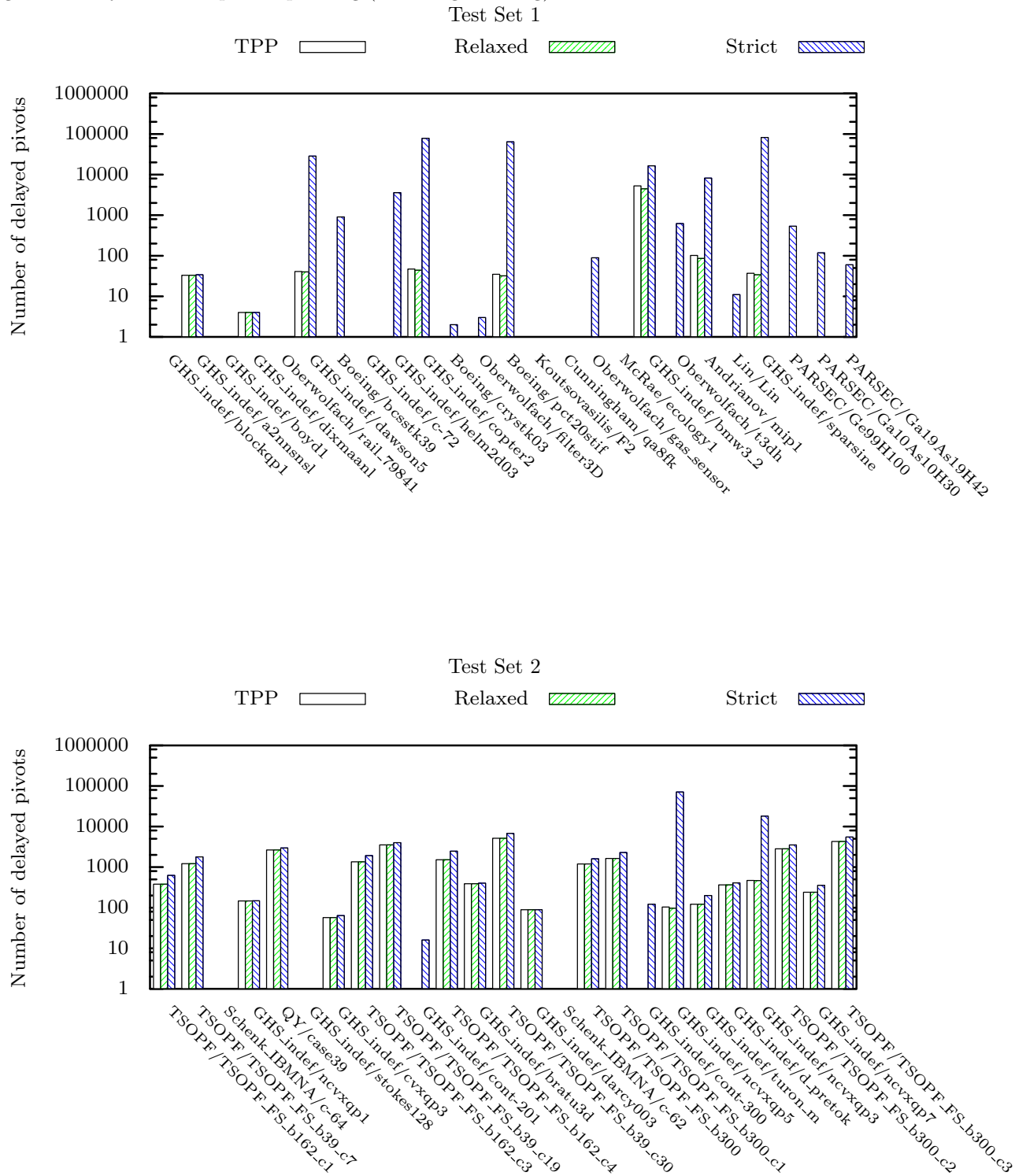
Test Set 2

TPP   Relaxed   Strict

Figure 7.6: The number of additional delays generated by compressed pivoting compared with the number generated by threshold partial pivoting (matching ordering).

Test Set 1

Test Set 2

8 Conclusions

Motivated by the need to devise algorithms that communicate as little as possible, even if they do slightly more arithmetic operations, we have presented two variants of a new pivoting algorithm for use within a sparse symmetric indefinite direct solver. Our proposed variants construct a compressed matrix using a tree reduction algorithm. We have shown that this results in better communication properties both practically and asymptotically than threshold partial pivoting, without compromising numerical robustness. Numerical tests demonstrate over a two times speedup for large problems. The strict compressed pivoting algorithm is provably stable but at the cost of potentially generating significantly more delayed pivots than threshold partial pivoting. An alternative relaxed compressed pivoting algorithm avoids this problem, but may not be stable on pathological examples. Nonetheless, it is shown that, in combination with appropriate scaling and ordering algorithms, it is stable in practice on even the most difficult of practical problems.

We note that many problems, if well scaled, do not require numerical pivoting, and in such cases the try-it-and-see approach suggested by Kim and Eijkhout [16] may be more appropriate. However for problems where unacceptable growth in the factor entries is detected, our new approach offers a fast alternative to identifying the minimal set of pivots that must be delayed during the factorization. Our future work is to develop software that uses such a technique and is targeted at manycore architectures such as GPUs or Intel's Xeon Phi.

Acknowledgements

References

[1] P. AMESTOY, I. DUFF, J.-Y. L'EXCELLENT, AND J. KOSTER, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Analysis and Applications, 23 (2001), pp. 15–41.

[2] C. ASHCRAFT, R. GRIMES, AND J. LEWIS, *Accurate symmetric indefinite linear equation solvers*, SIAM J. Matrix Analysis and Applications, 20 (1998), pp. 513–561.

[3] D. BARRON AND H. SWINNERTON-DYER, *Solution of simultaneous linear equations using magnetic-tape store*, The Computer J., 3 (1960), pp. 28–33.

[4] D. BECKER, M. BABOULIN, AND J. DONGARRA, *Reducing the amount of pivoting in symmetric indefinite systems*, Technical Report ICL-UT-11-06, University of Tennessee, 2011. Also INRIA Research Report 7621.

[5] J. BUNCH AND L. KAUFMANN, *Some stable methods for calculating inertia and solving symmetric linear systmes*, Mathematics of Computation, 31 (1977), pp. 163–179.

[6] T. DAVIS AND Y. HU, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011). Article 1, 25 pages.

[7] I. DUFF, A. ERISMAN, AND J. REID, *Direct Methods for Sparse Matrices*, Oxford University Press, 1986.

[8] I. DUFF AND J. KOSTER, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM J. Matrix Analysis and Applications, 22 (2001), pp. 973–996.

[9] I. DUFF AND S. PRALET, *Towards a stable static pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems*, SIAM J. Matrix Analysis and Applications, 29 (2007), pp. 1007–1024.

[10] L. Grigori, J. Demmel, and H. Xiang, *CALU: A communication optimal LU factorization algorithm*, SIAM J. Matrix Analysis and Applications, 32 (2011), pp. 1317–1350. Also LAPACK Working Note 266.

[11] A. Gupta, M. Joshi, and V. Kumar, *WSMP: A high-performance serial and parallel sparse linear solver*, Technical Report RC 22038 (98932), IBM T.J. Watson Research Center, 2001. `http://www.cs.umn.edu/~agupta/doc/wssmp-paper.ps`.

[12] J. Hogg and J. Scott, *An indefinite sparse direct solver for large problems on multicore machines*, Technical Report RAL-TR-2010-011, Rutherford Appleton Laboratory, 2010.

[13] ———, *HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems*, Technical Report RAL-TR-2011-024, Rutherford Appleton Laboratory, 2011.

[14] ———, *A study of pivoting strategies for tough sparse indefinite systems*, ACM Transactions on Mathematical Software, to appear (2013). See also Technical Report RAL-TR-2012-009, Rutherford Appleton Laboratory.

[15] HSL, *A collection of Fortran codes for large-scale scientific computation*, 2013. `http://www.hsl.rl.ac.uk/`.

[16] K. Kim and V. Eijkhout, *A parallel sparse direct solver via hierarchical dag scheduling*, Technical Report TR-12-05, Texas Advanced Computing Centre, 2012.

[17] X. Li and J. Demmel, *Making sparse Gaussian elimination scalable by static pivoting*, in Proceedings of the 1998 ACM/IEEE conference on Supercomputing, IEEE Computer Society, 1998, pp. 1–17.

[18] J. Reid and J. Scott, *An efficient out-of-core sparse symmetric indefinite direct solver*, Technical Report RAL-TR-2008-024, Rutherford Appleton Laboratory, 2008.

[19] ———, *Partial factorization of a dense symmetric indefinite matrix*, ACM Transactions on Mathematical Software, 38 (2011).

[20] O. Schenk and K. Gärtner, *On fast factorization pivoting methods for symmetric indefinite systems*, Electronic Transactions on Numerical Analysis, 23 (2006), pp. 158–179.

[21] O. Schenk, A. Wächter, and M. Hagemann, *Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization*, Computer Optimization and Applications, 36 (2007), pp. 321–341.

[22] D. Sorenson, *Analysis of pairwise pivoting in Gaussian elimination*, IEEE Transactions on Computers, C-34 (1985), pp. 274–278.

[23] L. Trefethhen and R. Schreiber, *Average-case stability of Gaussian elimination*, SIAM J. Matrix Analysis and Applications, 11 (1990), pp. 335–360.

A Communication analysis

A.1 General results

A.1.1 Serial factorization

The serial factorization of a $n \times p$ supernodal matrix forms the basis of much of our analysis of the parallel algorithms. To simplify our analysis, we will assume henceforth that all 2×2 pivots are accepted immediately and hence there is no need to apply permutations.

For each of $p/2$ pivots, the serial factorization performs the following steps. Counts are given for the i-th pivot.

- Find column maxima. 2 columns each with $(n - 2i)$ entries below the pivot require $2(n - 2i - 1)$ operations.

- Test pivot for acceptability. 18 operations.

- Apply (inverse of) pivot to 2 columns. 4 operations per row for total of $4(n - 2i)$ operations.

- Update the $(p - 2i)$ columns to the right of pivotal columns. Because of trapezoidal nature of the supernodal contribution to the factors, the number of entries to update is

$$(n - p)(p - 2i) + \sum_{j=1}^{p-2i} j = \frac{1}{2}(p - 2i)(2n - p - 2i + 1).$$

Each entry updated requires two fused multiply-add operations (one for each column of the pivot).

Summing across all pivots, we obtain the following operation count.

$$
\begin{aligned}
\mathrm{TPP}_{\mathrm{ops}}(n, p) &= \sum_{i=1}^{p/2} [2(n - 2i - 1) + 18 + 4(n - 2i) + (2n - p - 2i + 1)] \\
&= (16 + p - p^2 + 6n + 2np)\frac{p}{2} + (-14 - 4n)\sum_{i=1}^{p/2} i + 4\sum_{i=1}^{p/2} i^2 \\
&= (16 + p - p^2 + 6n + 2np)\frac{p}{2} + \frac{1}{2}(-14 - 4n)\frac{p}{2}(\frac{p}{2} + 1) + 4\frac{1}{6}\frac{p}{2}(\frac{p}{2} + 1)(p + 1) \\
&= \frac{29}{6}p - \frac{3}{4}p^2 - \frac{1}{3}p^3 + 2np + \frac{1}{2}np^2.
\end{aligned}
$$

A.1.2 Reduction on a tree

To perform the communication analysis it is necessary to have a model for reduction. Consider performing simultaneous reduction of k values on a binary tree, where only a single processor needs the final result. A binary tree across P processors has $1 + \log_2 P$ levels and therefore requires $\log_2 P$ messages to be sent. At each non-leaf node of the tree, k comparison operations performed. Given that there are P leaf nodes, the number of non-leaf nodes is

$$\sum_{i=1}^{\log_2 P} P2^{-i} = (1 - 2^{-\log_2 P})P = (P - 1).$$

Each non-leaf node has $2k$ words of information communicated to it. Hence,

$$
\begin{aligned}
\mathrm{Red}_{\mathrm{ops}}(k) &= (P - 1)k \\
\mathrm{Red}_{\mathrm{msg}}(k) &= \log_2 P \\
\mathrm{Red}_{\mathrm{bw}}(k) &= 2(P - 1)k.
\end{aligned}
$$

A.2 Threshold Partial Pivoting

We proceed to calculate theoretical bounds on the communication and computation for Variants A and B described in Section 2.1.

To simplify the analysis for Variant A, we assume that the first p rows (A_{11}) all reside on a single processor. Recall that for Variant B, all processors are sent a copy of these rows at the start (1 message using $\frac{1}{2}(P-1)p(p+1)$ words).

The significant differences from the serial factorization are in finding the column maxima (both variants) and communicating the chosen pivot (Variant A) or updating the local copy of A_{11} (Variant B).

In finding the column maxima for a single pivot, $2(P-1)$ comparisons are replaced by a global reduction of two values (one for each column). This generates no extra operations, but does generate an extra $\log_2 P$ messages containing a total of $4(P-1)$ words for each pivot.

For Variant A, the pivot and first $(p-2i)$ rows of the pivot columns must be communicated to other processors. This requires no additional operations, but requires 1 message and $2P(p-2i)+3$ words for pivot i $(1 \leq i \leq p/2)$. Hence,

$$
\begin{aligned}
\text{TPP}^A_{\text{ops}}(n,p) &= \text{TPP}_{\text{ops}}(n,p) \\
&= O(p^3 + np^2) \\
\text{TPP}^A_{\text{msgs}}(n,p) &= p + \frac{1}{2}p\log_2 P \\
&= O(p\log P) \\
\text{TPP}^A_{\text{bw}}(n,p) &= \sum_{i=1}^{p/2}[4(P-1)+2P(p-2i)+3] \\
&= (-\frac{1}{2}+2P+Pp)p - 2P(p/2)(p/2+1) \\
&= -\frac{1}{2}p + \frac{1}{2}Pp(p+2) \\
&= O(Pp^2).
\end{aligned}
$$

For Variant B, the leading $p \times p$ submatrix must be updated on every processor, incurring an additional $(P-1)\text{TPP}_{\text{ops}}(p,p)$ operations. This gives

$$
\begin{aligned}
\text{TPP}^B_{\text{ops}}(n,p) &= \text{TPP}_{\text{ops}}(n,p) + (P-1)\text{TPP}_{\text{ops}}(p,p) \\
&= \text{TPP}_{\text{ops}}(n,p) + (P-1)(\frac{29}{6}p + \frac{5}{4}p^2 + \frac{1}{6}p^3) \\
&= O(p^3 + np^2 + Pp^3) \\
\text{TPP}^B_{\text{msgs}}(n,p) &= 1 + \frac{1}{2}p\log_2 P \\
&= O(p\log P) \\
\text{TPP}^B_{\text{bw}}(n,p) &= \frac{1}{2}(P-1)p(p+1) + \sum_{i=1}^{p/2}[4(P-1)] \\
&= \frac{1}{2}(P-1)p(p+1) + 2(P-1)p \\
&= -\frac{1}{2}p(3-p) + \frac{1}{2}Pp(p+5) \\
&= O(Pp^2).
\end{aligned}
$$

A.3 Restricted Pivoting

To simplify the analysis, we again assume A_{11} resides on a single processor. First, A_{11} is factorized using serial threshold partial pivoting that requires no communication and has an operation count

$$\mathrm{TPP_{ops}}(p,p) = \frac{29}{6}p + \frac{5}{4}p^2 + \frac{1}{6}p^3.$$

The factor L_{11} is then communicated to all other processors, requiring 1 message per processor and $\frac{1}{2}(P-1)p(p+1)$ words of bandwidth. Each processor then applies this matrix to its own data. The number of operations required is

$$(n-p)\sum_{i=1}^{p/2}[4 + 2(p-2i)] = (n-p)(2p + p^2 - 2\frac{p}{2}(\frac{p}{2}+1))$$

$$= \frac{1}{2}(n-p)p(2+p).$$

Overall, we thus have:

$$\begin{aligned}
\mathrm{Restrict_{ops}}(n,p) &= \frac{29}{6}p + \frac{5}{4}p^2 + \frac{1}{6}p^3 + \frac{1}{2}(n-p)p(2+p) \\
&= \frac{29}{6}p + \frac{1}{4}p^2 - \frac{1}{3}p^3 + np + \frac{1}{2}np^2 \\
&= \mathrm{TPP_{ops}}(n,p) - p(n-p) \\
&= O(p^3 + np^2) \\
\mathrm{Restrict_{msg}}(n,p) &= 1 \\
\mathrm{Restrict_{bw}}(n,p) &= -\frac{1}{2}p(p+1) + \frac{1}{2}Pp(p+1).
\end{aligned}$$

A.4 Compressed Pivoting

Analysis of compressed pivoting (Section 4) follows that of restricted pivoting, except for the treatment of the leading submatrix A_{11}. The compressed matrix C is first assembled. In the strict algorithm, each processor scans the rows assigned to it and places them in the relevant set J_i. For each of $(n-p)$ rows this requires p absolute value operations and $(p-1)$ comparisons. Assuming the compressed matrix is initialized to zero, filling it requires one additional comparison for each of the $(n-p)p$ matrix entries. A reduction on p^2 values is then performed. In the relaxed algorithm, each processor scans each column of the rows assigned to it to find the largest local entry and compares that to the current largest for that row. This process requires a total of $(n-p)$ absolute value operations and $(n-p-P)+P$ operations per column. A modified reduction is then performed that uses $\mathrm{Red_{ops}}(p)$ operations but $\mathrm{Red_{bw}}(p^2)$ words of bandwidth. We summarise these results in the following table.

Table 1.2: Cost for construction of compressed matrix C.

	operations	messages	bandwidth
Strict	$(n-p)(3p-1) + (P-1)p^2$	$\log_2 P$	$2(P-1)p^2$
Relaxed	$(n-p)(2p) + (P-1)p$	$\log_2 P$	$2(P-1)p^2$

Having constructed C, the modified factorization is performed. The strict algorithm requires additional operations to work with the absolute value update operation on the $p \times p$ matrix C. This can be done

efficiently by storing an extra copy of A_{11} with the absolute value operation applied. The number of operations required is

$$\text{TPP}_{\text{ops}}(2p, p) + \frac{1}{2}p(p+1) = \frac{32}{6}p + \frac{15}{4}p^2 + \frac{2}{3}p^3.$$

As the relaxed algorithm uses an unmodified TPP algorithm, the number of operations it requires is

$$\text{TPP}_{\text{ops}}(2p, p) = \frac{29}{6}p + \frac{13}{4}p^2 + \frac{2}{3}p^3.$$

Summing with the application of L_{11} to A_{21}, we obtain the following counts,

$$
\begin{aligned}
\text{Compressed}_{\text{ops}}^{WC}(n, p) &= (n-p)(3p-1) + (P-1)p^2 + \frac{32}{6}p + \frac{15}{4}p^2 + \frac{2}{3}p^3 + \frac{1}{2}(n-p)p(2+p) \\
&= Pp^2 + \frac{19}{3}p - \frac{5}{4}p^2 + \frac{1}{6}p^3 - n + 4np + \frac{1}{2}np^2 \\
&= \text{TPP}_{\text{ops}}(n, p) + \frac{1}{2}p((p-1)p + 3) + n(2p-1) + Pp^2 \\
&= O(Pp^2 + p^3 + np^2) \\
\text{Compressed}_{\text{ops}}^{AC}(n, p) &= (n-p)(2p) + (P-1)p + \frac{29}{6}p + \frac{13}{4}p^2 + \frac{2}{3}p^3 + \frac{1}{2}(n-p)p(2+p) \\
&= Pp + \frac{23}{6}p + \frac{1}{4}p^2 + \frac{1}{6}p^3 + 3np + \frac{1}{2}np^2 \\
&= \text{TPP}_{\text{ops}}(n, p) + \frac{1}{2}p((p+2)p - 2) + (n+P)p \\
&= O(Pp + p^3 + np^2).
\end{aligned}
$$

The strict and relaxed algorithms have the same communication pattern. Summing the compressed matrix construction with the later distribution of L_{11} yields

$$
\begin{aligned}
\text{Compressed}_{\text{msg}}(n, p) &= 1 + \log_2 P \\
&= O(\log P) \\
\text{Compressed}_{\text{bw}}(n, p) &= 2(P-1)p^2 + \frac{1}{2}(P-1)p(p+1) \\
&= -\frac{1}{2}p(5p+1) + \frac{1}{2}Pp(5p+1) \\
&= O(Pp^2).
\end{aligned}
$$