Accelerated methods for performing the LDLT decomposition

Peter E. Strazdins[*]

(Received 7 August 2000)

Abstract

This paper describes the design, implementation and performance of parallel direct dense symmetric-indefinite matrix factorisation algorithms. These algorithms use the Bunch-Kaufman diagonal pivoting method. The starting point is numerically identical to LAPACK `_sytrf()` algorithm, but out-performs `zsytrf()` by $\approx 15\%$ for large matrices on the UltraSPARC family of processors. The first variant

[*]Department of Computer Science, Australian National University, Canberra, ACT 0200, Australia. mailto:peter@cs.anu.edu.au

[0]See http://anziamj.austms.org.au/V42/CTAC99/Stra for this article and ancillary services, © Austral. Mathematical Soc. 2000. Published 27 Nov 2000.

reduces symmetric interchanges, particularly important for parallel implementation, by taking into account the growth attained by any preceding columns that did not require any interchanges. However, it achieves the same growth bound.

The second variant uses a *lookahead* technique with heuristic methods to predict whether interchanges are required over the next block column; if so, the block column can be eliminated using modified Cholesky methods, which can yield both computational and communication advantages.

These algorithms yield best performance gains on 'weakly indefinite' matrices (i.e. those which have generally large diagonal elements), which often arise from electro-magnetic field analysis applications. On UltraSPARC processors, the first variant generally achieves a 1–2% performance gain; the second is faster still for large matrices by 2% for complex double precision and 6% for double precision.

However, larger performance gains are observed on distributed memory machines, where symmetric interchanges are relatively more expensive. On a 16 node 300 MHz UltraSPARC-based Fujitsu AP3000, the first variant achieved a 10-15% improvement for small-moderate sized matrices, decreasing to 7% for large matrices. For $N = 10000$, it achieved a sustained speed of 5.6 GFLOPs and a parallel speedup of 12.8.

Contents

1 Introduction

Large symmetric indefinite systems of equations arise in many applications, including incompressible flow computations and optimisation of linear and non-linear programs. They also arise in electro-magnetic field analysis, such as in the ACCUFIELD application [8]. Here, these systems are also complex, can be very large (e.g. $N \approx 30000$), and often the diagonal elements are relatively large. For such *weakly indefinite* systems, i.e. transformations appropriate to definite systems can be applied to eliminate most columns of these matrices without sacrificing numerical stability, important computational advantages can be gained.

Stable algorithms for solving $N \times N$ symmetric indefinite systems and yet exploit symmetry to have only $\frac{N^3}{3} + O(N^2)$ floating point operations are well known (see [4] and the references within). While several performance evaluations of variants of these algorithms have been given [1, 6, 7, 2], all but [6] consider only uniprocessor implementations, and [6] only considers parallelization on a small-scale shared memory machine.

In this paper, we describe how to derive variants of the Bunch-Kaufman diagonal pivoting method to yield improved performance, especially on parallel platforms.

The parallel routines are coded entirely in terms of the DBLAS Distributed BLAS Library [10, 12, 11], which is a portable version of parallel BLAS. It has been used to implement very efficient parallel matrix factorization applications using various techniques [11]. The use of the DBLAS has enabled rapid development and prototyping of several variants of the Bunch-Kaufman algorithm, while enabling high reliability and performance from its highly tested and optimized components.

The Fujitsu AP3000 [5] is a distributed memory multicomputer, comprised of RISC scalar processors (UltraSPARC) with a deep memory hierarchy (having a 16KB top-level data cache and a 1MB 2nd-level cache, both direct-mapped, and a 64-entry TLB). It has communication networks with characteristics shared by most other state-of-the-art distributed memory computers.

The main original contributions of this paper are as follows: it provides an analysis of the issues of symmetric indefinite factorizations for distributed memory platforms; and it develops and evaluates two new variants of the diagonal pivoting algorithm that yield superior serial and especially parallel performance, discussing issues required for their efficient parallelization.

2 Diagonal Pivoting Methods

The Bunch-Kaufman method performs the decomposition $A = LDL^T$, where L is an $N \times N$ lower triangular matrix with a unit diagonal, and D is a block diagonal matrix with either 1×1 or 2×2 sub-blocks [4]. A 2×2 sub-block indicates a 2×2 pivot was required for the stable elimination of the corresponding columns; the corresponding sub-diagonal element of L will be 0. In a practical implementation of this method, A can then be overwritten by L and D, with a 'pivot vector' recording any symmetric interchanges (including the position of the 2×2 pivots) [4, 1, 6].

For the sake of brevity, the notations x' $(\tilde{x})$, for an integer expression x, is a shorthand for $x + 1$ $(x - 1)$.

In the elimination of column j, four cases can arise with the Bunch-Kaufman method:

D1 $|A_{jj}| \geq \alpha |A_{ij}|$, where $j < i < N$ and $|A_{i,j}| = \max_{k=j'}^{\tilde{N}} |A_{k,j}|$. Here, a 1×1 pivot from $A_{j,j}$ will be stable; no symmetric interchange is required.

D2 the conditions for D1 and D4 do not hold. Here, $A_{j,j}$ is used as a 1×1 pivot, and no symmetric interchange is required.

D3 A 1×1 pivot from A_{ii} will be stable. Here, a symmetric interchange with row / columns i and j must be performed.

D4 A 2×2 pivot using columns j and i will be stable. Here, a symmetric interchange with row / columns i and $j + 1 = j'$ must be performed; however, both columns are eliminated in this step.

α is a tuning constant for the algorithm; it can be shown that $\alpha = \frac{1+\sqrt{17}}{8}$ maximizes stability of this algorithm [4]. For definite systems, only case D1 is needed; case D3 is also needed for semi-definite systems, and case D4 is needed for indefinite systems.

Case D2 exists primarily to avoid a situation where case D4 might be unstable. By stability, it is meant that the growth of the trailing sub-matrix (A' in Figure 1) is bounded; however, due to cases D2 and D4 there is no guarantee that the growth of L is bounded [2]; recently a *bounded Bunch-Kaufman* algorithm has been presented which overcomes this problem [2].

It has been argued that due to the sufficiently high stability for most practical purposes, the choice between diagonal pivoting methods and their main alternative, tridiagonal reduction method [4], is essentially an issue of performance [9, 13]. For parallel implementation, it has been identified that the *symmetric interchanges* (e.g. interchange of rows *and* columns i and j, see Figure 1(a)), required to keep such algorithms stable, involves high communication costs (relative to LU and LLT factorizations); thus the diagonal pivoting methods, which afford a reduction in the number of these (especially for weakly indefinite matrices) should be strongly favoured [9, 13].

It was also noted there that while several variants of the diagonal pivoting method have since been proposed, the LAPACK implementation of the Bunch-Kaufman method [1] has proven to be very competitive in terms of performance over a range of platforms. Thus, this will form the starting point for our parallel implementation.

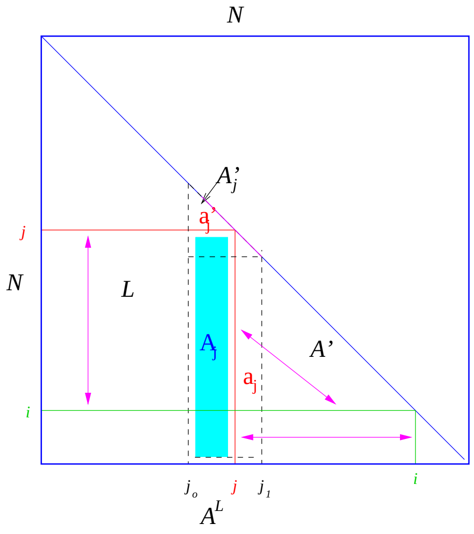
The choice between the bounded and original Bunch-Kaufman algorithm deserves some treatment. While the bounded algorithm offers better stability, empirical and analytical studies show that it requires on average at least 2.5 column searches every time the test for case D1 fails [2]. Furthermore, empirical studies on random matrices have shown that the average number of symmetric interchanges of the bounded algorithm is ≈ 1.7 times greater [2]. Unless such stability is required for a particular application, this extra overhead favours the original algorithm.

2.1 The LAPACK Diagonal Pivoting Algorithm

In this section, we describe a diagonal pivoting algorithm, essentially a simplification of that in LAPACK `_sytrf()`, which is publicly available from NetLib [3]. A concise matrix-notation description of this algorithm is can be found in [9]; an informal description of the partial factorization is given in Figure 1. Here, the elements of the symmetric matrix A are stored in its lower triangular half, with A^L (the dashed trapezoid) being the panel currently being factored. Note that w_j and w'_j refer to the jth column and row of W. For the sake of simplicity, it is assumed that the input matrix is invertible.

At this point, we can begin to analyse the relative costs of each of the cases D1–D4. Case D1 is the most efficient; case D2, while having the same outcome, requires half a symmetric interchange and wastes the second matrix-vector multiply. Case D3 is less efficient still, as it similarly wastes the first matrix-vector multiply, and requires the symmetric interchange to be completed. Case D4 is between cases D1 and D2 in terms of efficiency, for although it requires a symmetric interchange, it eliminates 2 columns. Thus, unlike cases D2 and D3, it involves no redundant matrix-vector multiplies.

As described in more detail in [9], LDLT has the following performance advantages over LLT factorization: 1(a) it must form A^L via level 2 compu-



```
create new matrix $W$ aligned with $A^L$
for each $j$ in current panel
    $w_j \leftarrow (w'_j)^T A_j + a_j$;
    determine the pos. $i$ of the max. in $w_j$
    if case D1 does not apply:
        (partial) interchange $j+1$ with $i$
        $w_{j+1} \leftarrow (w'_{j+1})^T A_j + a_{j+1}$
        find max. of $w_{j+1}$
        if D3-D4 applies, complete interchange
    if case D4 applies
        $D_j = \begin{pmatrix} A_{j,j} & A_{j+1,j} \\ A_{j+1,j} & A_{j+1,j+1} \end{pmatrix}$
        $(a_j, a_{j+1}) \leftarrow (w_j, w_{j+1}) D_j^{-1}$
        skip column $j+1$
    else
        $a_j \leftarrow w_j / A_{j,j}$
$A'$-=$A^L W^T$
```

(a) matrix components (b) LAPACK algorithm

FIGURE 1: Partial Factorization of an $N \times N$ symmetric indefinite matrix A

tations, 1(b) pipelined communication cannot be used for horizontal broadcasts, 1(c) N column searches for the maximum elements are required (with immediate broadcast to *all* processors), 1(d) advanced load balancing techniques such as *pipelining with lookahead* [11] cannot be used, and 2(a) W must be explicitly transposed before its vertical broadcast. The first 4 arise from the *possibility* of a symmetric interchange occurring in the formation of A^L.

In turn, it has been argued that LLT will generally have considerably lower parallel speed (in FLOPs rate) than LU [9]; thus there will be a value N_{LU} for which the general LU factorization will be faster if $N < N_{\mathrm{LU}}$, despite requiring twice as many floating point operations.

2.2 Parallelizing the Diagonal Pivoting Method

The LAPACK LDLT algorithm, based on BLAS operations with a high fraction of level-3 computations due to the blocking factor or panel width $\omega > 1$, has been shown to be efficient on memory hierarchy uniprocessors [1, 6, 7, 2]. Thus, in principle, as other processor's memory can be regarded as an extra level of the memory hierarchy in the distributed memory context, the algorithm depicted by Figure 1(b) should have a straightforward parallelization that is also reasonably efficient. However, several modifications and optimizations can still be performed.

We will consider the $r \times s$ block-cyclic matrix distribution over a $P \times Q$

logical processor grid [10], where, for an $N \times N$ global matrix A, block (i, j) of A will be on processor $(i \bmod P, j \bmod Q)$. For this distribution, two established techniques can be used to parallelize this algorithm: *storage blocking*, where $\omega = r = s$, and *algorithmic blocking*, where $\omega > r = s \approx 1$. The latter has load balance advantages, at the expense of extra communication startup costs; it has been shown to yield better performance across platforms with relatively low communication costs [11]. Our implementation encompasses both techniques.

In [9], various optimizations to reduce communication startup costs in the LDLT factorization and backsolve computations are described. These include combining the communications for individual array elements with other communications, and, unlike in the LAPACK algorithm, completing the row swaps in L to afford a faster serial and parallel backsolve computation that uses only standard (parallel) BLAS components.

3 Minimizing Symmetric Interchanges

As explained in Section 2.1, minimizing the amount of symmetric interchanges (while keeping the algorithm stable) has potentially large gains in the parallel algorithm performance.

One method of achieving this is implementing a key idea in the algorithm in [6]. This algorithm was largely motivated by the requirements of band matrices, where the minimization of interchanges helps preserve the structure of these matrices [6]. Let k be the current column of A^L to be eliminated, and let column $k-p$, $0 < p \leq k$ be the last column not eliminated by case D1. Let λ_i be the absolute value of the maximum element of the ith subdiagonal. Then the condition for determining case D1 can be relaxed to:

$$\mu_k = \prod_{i=k-p+1}^{k} \left(1 + \frac{|W_{i,i}|}{\lambda_i}\right) \leq \left(1 + \frac{1}{\alpha}\right)^p \tag{1}$$

This is stable in the sense that the overall growth of A' from the block of p 1×1 pivots still remains within its bounds [6]. Intuitively, this can be thought of as the existence of large diagonal elements in preceding columns reducing the growth bounds on A' sufficiently to compensate for a smaller current diagonal element.

The implementation of this idea is somewhat different here however. The algorithm in [6] is based on a different (3-case) variant of the diagonal pivoting method, and furthermore uses an *a priori* growth bound instead (which is necessarily more conservative, see Section 4). There, the value of p becomes reset whenever the target blocking factor ω is reached, *or* a symmetric interchange is required. This has the undesirable consequences of the blocking factor p often falling short of the target blocking factor ω [6], resulting in a reduction in computational performance, and, in a distributed memory implementation, it would compromise the advantages of using storage blocking, as often the panel A^L would be straddling a storage block boundary.

Our implementation limits p to the range $0 \leq p < p_{\max}$, which is necessary to avoid overflow in μ_k. This can be efficiently achieved by storing the values of $1+\frac{|W_{i,i}|}{\lambda_i}$ in a circular queue of size $p_{\max}$. Thus, μ_k can include contributions from columns in previous blocks. Furthermore, the optimal blocking factor is always met regardless of whether Equation 1 is.

A potential problem with this *reduced interchange method* is that, compared with the original Bunch-Kaufman method, they allow increased growth in L_j by a factor of $|\frac{a_{j,j}}{\lambda_j}| \approx \alpha^p$ [2]. A compromise, which we call the *guarded reduced interchange* variant, would to disallow case D1 when $|\frac{a_{j,j}}{\lambda_j}| > \alpha^{p_0}$, where $p_0 \ll p_{\max}$, i.e. $p_0 = 5$, even when Equation 1 is satisfied.

Table 1 lists the normalized residual (using a random RHS vector with elements from the unit circle) for these methods for sample ACCUFIELD matrices. The normalized residual is calculated the same way as in LAPACK test programs [3]; ideally, an accurate algorithm will produce residuals of less than unity, although in practice it may occasionally exceed unity (especially for small matrices) without implying a significant loss in accuracy. It also shows the fraction f of columns eliminated by cases D2–D4. With the typically large diagonal elements of these matrices, generally $f < 0.1$.

Figure 2 extends this study to simulated matrices of the form $A = A' + \beta I$, where A' has random elements from the unit circle and $0 \leq \beta \leq 10$. These represent the averaged values of the residual (and f) for 10 such matrices as functions of the diagonal bias β. In terms of the pivot distribution f, the range $5 \leq \beta \leq 7$ corresponds to Table 1. In terms of accuracy, the reduced interchange method has a residual generally within twice that of

TABLE 1: Comparison of residual (and f) for diagonal pivoting methods for ACCUFIELD matrices

N :	53		161		1601	
original:	1	(.04)	.01	(.06)	.01	(.07)
reduced:	2	(0)	.02	(.07)	.02	(.01)
guarded:	.02	(.05)	.02	(.02)	1.0	(.02)

the original method, except for the range $3 \leq \beta \leq 5$, which coincides with the largest absolute reduction in f (note however all residuals are all within their threshold here). The guarded method has comparable accuracy to the original, and both of the new methods significantly reduce f for $\beta > 1$; in particular $f \approx 0$ for $\beta > 5$.

In terms of stability, this scheme is no worse than the original Bunch-Kaufman method in the sense that it attains the same growth bound in A'.

4 Lookahead Techniques

As discussed at the end of Section 2.1, LLT factorization has several inherent performance advantages over LDLT factorization. However, for weakly indefinite matrices, which we assume for this section, it should be possible to

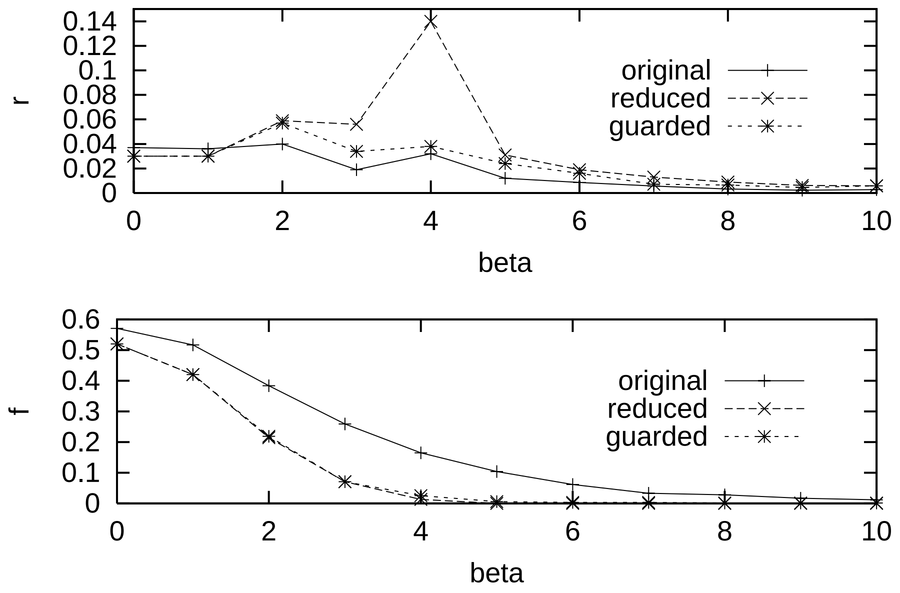


Figure 2: Averaged residual and pivot distribution for simulated 500×500 matrices

design an LDLT solver whose performance approaches that of an LLT solver.

The key idea is to *lookahead* over the next block of ω columns in A^L and search for their (current) maximum elements λ'_k, and then predict the *a priori* element growth for the 1st k diagonals of the block being used as the pivots [6].

$$\mu^{\mathrm{p}}_k = \mu^{\mathrm{p}}_{k-1}\left(1 + \frac{\lambda^{\mathrm{p}}_k}{|W_{k,k}|}\right) \tag{2}$$

$$\lambda^{\mathrm{p}}_k = \lambda'_k + \sum_{i=0}^{k-1} |A_{k,i}|\lambda^{\mathrm{p}}_i \tag{3}$$

Note that the final values of the diagonal elements are required by Equation 2; this implies that the triangular part of A^L must be computed first; this can be done using a modified Cholesky level-2 factorisation (which also produces the corresponding triangular part of W). Equation 3 thus gives an upper bound on the growth of the maximum due to updates from the previous $k-1$ columns [6].

The algorithm of [6] terminates the search whenever this bound is exceeded, but suggests that a search over all columns would be efficient provided this occurs infrequently. In the distributed memory setting, the simultaneous search over ω columns reduces the communication startup costs of the searches by a factor of ω, in other words, reducing startup costs to $O(\frac{N}{\omega})$, comparable to LLT.

Furthermore, referring again to Figure 1 with $\omega^{\mathrm{p}} = j - j_0$ and W'_j being the triangular part of W corresponding to A'_j (and similarly W_j corresponding to A_j), the first ω^{p} columns of A_j can be eliminated by applying the level-3 triangular matrix update$W_j \leftarrow A_j(W'_j)^{-T}$. W_j can be horizontally and then vertically broadcast as for LLT; using then $A_j = W_j D_j^{-1}$, where D_j is the diagonal matrix corresponding to A'_j, A_j can then be reconstructed in all processors.

Thus, the disadvantages 1(a), 1(b), 1(c) and 2(a) of Section 2.1 may be overcome provided on average $\omega^{\mathrm{p}} \approx \omega$. Apart from 1(d), the only significant performance disadvantages over an LLT algorithm would be the memory accesses in extra level-1 operations (in the column searches, and in the reconstruction of A_j).

4.1 Details of an Effective Lookahead Algorithm

While the potential advantages of this method are many, devising an algorithm that can realize them consistently and out-perform the previous variant turns out to be non-trivial. In particular, it is desirable to exploit the advantages of calculating the accumulated growth in previous columns, whether lookahead was applied to them or not.

The first problem is that for most weakly indefinite matrices, $\mu^{\mathrm{p}}_{\omega^{\mathrm{p}}}$ exceeds $\mu_{\omega^{\mathrm{p}}}$ for large ω^{p}, say $\omega^{\mathrm{p}} \geq \omega = 44$, often by one or two orders of magnitude. Table 2 gives the average block sizes produced by using Equation 2 combined

Table 2: Average size of block columns using *a priori* growth bounds for complex simulated matrices of various β for $N = 528$ and a target $\omega = 44$

β	4	6	8	10	15	20	25	30
$\bar{\omega}^{\mathrm{p}}$	2	2	2	4	6	16	32	37
$\bar{\omega}^{\mathrm{p}\prime}$	10	13	18	27	37	37	40	40

with using the method of Section 3. The latter can potentially improve ω^p as it allows growth 'credits' from previous blocks to be used. Even so, the averaged block size $\bar{\omega}^p$ fell far below the target blocking factor of $\omega = 44$, except for larger β.

Note that for this range of β, the pivoting ratio $f \approx 0$, indicating that $\bar{\omega}^{\mathrm{p}} \approx \omega$ is in principle possible.

The $\bar{\omega}^p$ achieved here is insufficient for attaining performance gains by using the lookahead method for $\beta < 25$. This is because the *a priori* bounds are too conservative to achieve large blocking factors for a broad range of weakly indefinite matrices.

A first attempt to remedy this is, after the elimination of A_j, to recompute $\mu_{0:\omega^{\mathrm{p}}}$ by performing a second column search over $(A'_j|A_j)$. After performing the block elimination of A_j, column j must be eliminated using the algorithm of Figure 1. Surprisingly often, it turns out that it will be eliminated by cases D1 or D2, which means that lookahead can be applied over the next

$\omega_1^{\mathrm{p}} \leq \omega - \omega^{\mathrm{p}} - 1$ columns. In this case, we can take advantage of the fact that the exact values of $\lambda_{0..\omega^{\mathrm{p}}}$ are now available to recompute the bounds on the growths of the column maximums less conservatively:

$$\lambda_k^{\mathrm{p}} = \lambda_k' + \sum_{i=0}^{\omega^{\mathrm{p}}} |A_{k,i}|\lambda_i + \sum_{i=\omega^{\mathrm{p}}+1}^{\omega} |A_{k,i}|\lambda_i^{\mathrm{p}} \qquad (4)$$

If this is deemed to be stable, the columns to the right of j must be updated by $(A_{j+1})_{\omega_1^{\mathrm{p}}:,:}(W_{j+1})^T_{\omega_1^{\mathrm{p}}:\omega-1,:}$ before elimination.

This resulted in an average blocking factor $\bar{\omega^{\mathrm{p}}}'$ more quickly approaching ω, as shown in Table 2. Even this is insufficient for improvement of performance on the ACCUFIELD matrices, which correspond to $5 \leq \beta \leq 10$.

This problem had to be overcome by using a *heuristic* estimate of the element growth; a workable choice is to use Equation 1 but using the values of the column maximums λ_k' *before* the updates for the current panel are applied. This is based on the observation that the ratio of the diagonal elements to their respective column maximums is likely to remain relatively stable over the updates. However, a copy of the original panel must be kept in case an unstable elimination is subsequently detected by the second column search, so that the panel factorisation can be restarted from that point[1].

The execution time required by computing Equations 2 and 3 would introduce a significant overhead if recomputed at too many columns of a panel.

[1]Preliminary experiments on matrices with low f indicate this is sufficiently rare not to imply a performance loss.

Furthermore, lookahead will only yield any performance advantage if ω^{p} is sufficiently large. Thus, whenever $\omega^{\mathrm{p}} < \omega^{\mathrm{p}}_{\mathrm{min}}$, our implementation does not apply lookahead but instead eliminates the next $\omega^{\mathrm{p}}_{\mathrm{min}}$ columns in the normal fashion. For the UltraSPARC, a value of $\omega^{\mathrm{p}}_{\mathrm{min}} = 16$ was found to be optimal.

Finally, in computing $A_j \leftarrow A_j(W'_j)^{-T}$, the standard parallel BLAS `TRSM()` routine was extended to also compute W_j, taking advantage of the fact that W_j is the value of A_j before the final scaling by the diagonal elements elements in W'_j. This avoided extra level-1 computations.

5 Performance

The components of our factorisation routine, the DBLAS parallel BLAS and the UltraSPARC serial BLAS, have been very highly optimised for the AP3000; for further details, see [9, 12]. The complex precision version of the routine is named `DZSYTRF()`.

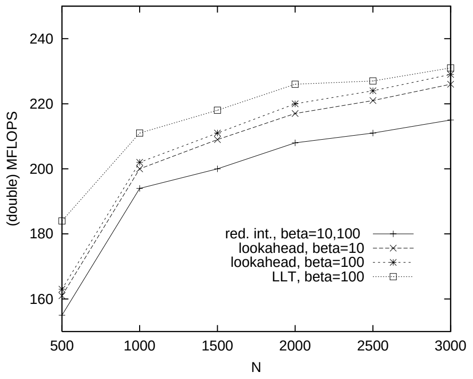
The DBLAS implementation of LDLT decomposition allows the grid size $P \times Q$, the storage block size r and the algorithmic blocking size ω to be run-time settable parameters. Thus, simply setting $\omega = r$ means that storage blocking will be used; the DBLAS routines then ensure all the communication savings from storage blocking then occur. Thus, given a matrix of size N and PQ processors, the optimum combination of these parameters can then be chosen. Here, MPI was used as the underlying communication library.

TABLE 3: LDLT solver performance in complex MFLOPS for ACCUFIELD matrices on a U300 ($\omega = 44$)

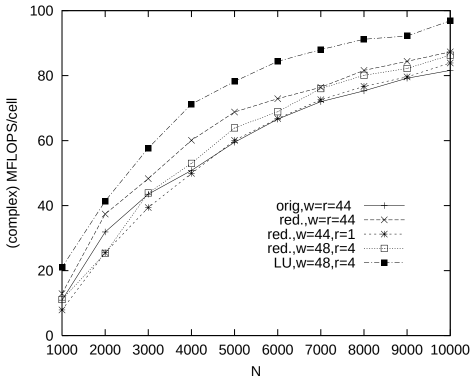
	`zsytrf()`	`DZSYTRF()`		
N		orig.	reduced	lookahead
161	80	73	74	76
1601	96	108	109	111

Table 3 compares the performance of the LDLT factorisation routines on a 300 MHz UltraSPARC II (U300). The optimum blocking factor for complex matrices was $\omega = 44$. See [9, 13] for a discussion of the results relative to LAPACK `zsytrf()`.

The above results belie the potential of the lookahead method for improving computational speed. A clearer improvement for double precision is observed because a larger blocking factor of $\omega = 64$ is required and there is a greater difference between matrix-vector multiply and matrix-matrix multiply speeds. Figure 3(a) shows some results comparing the lookahead version with the reduced interchange version. For $\beta = 10$, $f < 0.01$ for both versions; for $\beta = 100$, $f = 0$, i.e. the matrices were positive definite and an LLT algorithm could be used for comparison. Here, we see that the lookahead method did indeed approach LLT speed, gaining a 5–6% improvement over the entire range. Note that the residuals for the lookahead version were generally less than the reduced interchange version.



(a) double; 1×1, 170 MHz



(b) complex; 4×4, 300 MHz

FIGURE 3: LDLT performance for simulated $N \times N$ matrices on an AP3000

Figure 3(b) gives parallel factorisation performance for simulated matrices with $\beta = 7$. It was found that the pivoting ratio f increases significantly with N; e.g. at $N = 10000$, $f = 0.20$ for the original method, and $f = 0.05$ with reduced interchange.

Comparing with Table 3, it can be seen that communication overheads prevent efficiencies that are possible in the serial case. Comparing the plots for $\omega = r = 44$, we can see that much of this overhead is from the interchanges, with the reduced interchange version being faster by $\approx 10 - 15\%$ at the low-mid ranges, decreasing to $\approx 7\%$ at the upper range.

Comparing the plots for the reduced version, we see that storage blocking ($\omega = r = 44$) has a small but consistent advantage over algorithmic blocking ($\omega \approx 44$ with $r = 1, 4$). This is to be expected for small N, as algorithmic blocking incurs an extra $\lg_2 QN$ startups and an extra $\lg_2 Q\frac{N}{P}$ communication volume to broadcast horizontally a_j for the vector-matrix multiply. For moderate-large N, storage blocking has an unusual advantage on the AP3000: its larger messages in the broadcast of A^L and W^T have an effectively higher bandwidth.

However, for LU, and to a lesser extent LDLT without reduced interchanges, algorithmic blocking at $\omega = 48, r = 4$ slightly out-performed storage blocking for $N > 5000$. While LU achieves somewhat higher speeds for a given N, they are never greater than 2 (i.e. $N_{\mathrm{LU}} < 1000$ here). In other words, the LDLT factorization is quicker than LU in this range, with the residuals for LU being only marginally smaller.

6 Conclusions

Symmetric indefinite matrix factorization is an interesting computation where there is a trade-off in accuracy and performance. We have presented and developed variants of the diagonal pivoting method which yield improved performance, especially for 'weakly indefinite' systems where it is easier to obtain high accuracy. These variants are also useful for preserving the structure of banded matrices.

Our choice of algorithms was guided by a need for methods which afford a reduction of symmetric interchanges, due to their relatively high costs on distributed memory platforms. This lead to the adoption of the diagonal pivoting method, and the developments of variants which could afford further reductions. Our starting point was thus based on the LAPACK `_sytrf()` routine, but with its low-level components highly optimized. This already yielded a clear performance gain over `_sytrf()` for large matrices on the UltraSPARC family of processors; to our knowledge, this is the first time such a result has been demonstrated.

The first variant afforded a further reduction in symmetric interchanges by considering the accumulated growth of previous columns. While this yielded only modest performance improvements for serial computation, on state of the art distributed memory platforms such as the Fujitsu AP3000, the reduction in communication costs resulted in a 15–7% increase from small to large matrices.

The second variant achieved a similar reduction in symmetric interchanges but applied lookahead to predict where block columns could be stably eliminated using modified Cholesky methods. However, non-trivial issues had to be addressed before this variant could yield a consistent improvement over the first. This included the introduction of heuristic *a priori* growth estimates. The higher fraction of level 3 computation afforded by this method yielded a 6% performance increase for (nearly-) definite double precision matrices, even for large sizes, approaching LLT factorization speed. It also has considerable scope for reducing communication startup and volume costs.

Acknowledgements: The author thanks Mark Jones, Linda Kaufman and John Lewis for their insights into why tridiagonal-based methods have received so little development over recent years. Many thanks also to John Lewis for suggestions leading to the *guarded reduced interchange* method. Finally, the author would also like to thank the Fujitsu Parallel Research Computing Facilities for the use of their AP3000.

References

[1] C. Anderson and J. Dongarra. Evaluating block algorithm variants in LAPACK. In *Fourth SIAM Conference for Parallel Processing for Scientific Computing*, Chicago, December 1989. 6 pages. C1331, C1332, C1334, C1337

[2] Cleve Ashcraft, Roger G. Grimes, and John G. Lewis. Accurate symmetric indefinite linear equation solvers. *Simax*, 1999. 49 pages. C1331, C1334, C1334, C1334, C1334, C1337, C1340

[3] C. Anderson et al. *LAPACK User's Guide*. SIAM Press, Philadelphia, 1992. C1335, C1340

[4] Gene Golub and Charles Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, second edition, 1989. C1331, C1332, C1332, C1333, C1334

[5] H. Ishihata, M. Takahashi, and H. Sato. Hardware of the AP3000 parallel server. *Fujitsu Scientific and Technical Journal*, 33(1):24–29, 1997. C1331

[6] Mark T. Jones and Merrell L. Patrick. Factoring symmetric indefinite matrices on high-performance architectures. *SIAM Journal on Matrix Analysis and Applications*, 12(3):273–283, July 1991. C1331, C1331, C1331, C1332, C1337, C1338, C1339, C1339, C1339, C1339, C1343, C1343, C1343

[7] Linda Kaufman. Computing the MDM^T decomposition. *ACM Transactions on Mathematical Software*, 21(4):476–489, December 1995. C1331, C1337

[8] Dr. Noro. Private communications, 1998–1999. C1330

[9] P. E. Strazdins. A dense complex symmetric indefinite solver for the Fujitsu AP3000. Technical Report TR-CS-99-01, Computer Science Dept, Australian National University, May 1999, `http://cs.anu.edu.au/techreports/1999/TR-CS-99-01.html`. C1334, C1334, C1335, C1335, C1337, C1338, C1347, C1348

[10] P.E. Strazdins. Reducing software overheads in parallel linear algebra libraries. In *The 4th Annual Australasian Conference on Parallel And Real-Time Systems*, pages 73–84, Newcastle Australia, September 1997. Springer. C1331, C1338

[11] P.E. Strazdins. Lookahead and algorithmic blocking techniques compared for parallel matrix factorization. In *PDCN'98: 10th International Conference on Parallel and Distributed Computing and Systems*, pages 291–297, Las Vegas, September 1998. IASTED. C1331, C1331, C1337, C1338

[12] Peter E. Strazdins. Transporting distributed BLAS to the Fujitsu AP3000 and VPP-300. In *Proceedings of the Eighth Parallel Computing Workshop*, pages 69–76, Singapore, September 1998. School

of Computing, National University of Singapore. paper P1-E. C1331, C1347

[13] Peter E. Strazdins. Parallelizing dense symmetric indefinite solvers. In *Proceedings of 6th Annual Australasian Conference on Parallel And Real-Time Systems*, pages 398–410, Melbourne, November 1999, Springer-Verlag. C1334, C1334, C1348