# Blocked Schur Algorithms for Computing the Matrix Square Root

Edvin Deadman[1], Nicholas J. Higham[2], and Rui Ralha[3]

[1] Numerical Algorithms Group
edvin.deadman@nag.co.uk
[2] University of Manchester
higham@maths.manchester.ac.uk
[3] University of Minho, Portugal
r_ralha@math.minho.pt

**Abstract.** The Schur method for computing a matrix square root reduces the matrix to the Schur triangular form and then computes a square root of the triangular matrix. We show that by using either standard blocking or recursive blocking the computation of the square root of the triangular matrix can be made rich in matrix multiplication. Numerical experiments making appropriate use of level 3 BLAS show significant speedups over the point algorithm, both in the square root phase and in the algorithm as a whole. In parallel implementations, recursive blocking is found to provide better performance than standard blocking when the parallelism comes only from threaded BLAS, but the reverse is true when parallelism is explicitly expressed using OpenMP. The excellent numerical stability of the point algorithm is shown to be preserved by blocking. These results are extended to the real Schur method. Blocking is also shown to be effective for multiplying triangular matrices.

## 1 Introduction

A square root of a matrix $A \in \mathbb{C}^{n\times n}$ is any matrix satisfying $X^2 = A$. Matrix square roots have many applications, including in Markov models of finance, the solution of differential equations and the computation of the polar decomposition and the matrix sign function [12].

A square root of a matrix (if one exists) is not unique. However, if $A$ has no eigenvalues on the closed negative real line then there is a unique *principal square root* $A^{1/2}$ whose eigenvalues all lie in the open right half-plane. This is the square root usually needed in practice. If $A$ is real, then so is $A^{1/2}$. For proofs of these facts and more on the theory of matrix square roots see [12].

The most numerically stable way of computing matrix square roots is via the Schur method of Björck and Hammarling [6]. The matrix $A$ is reduced to upper triangular form and a recurrence relation enables the square root of the triangular matrix to be computed a column or superdiagonal at a time. In §2 we show that the recurrence can be reorganized using a standard blocking scheme or recursive blocking in order to make it rich in matrix multiplications. We show experimentally that significant speedups result when level 3 BLAS are exploited in the implementation, with recursive blocking providing the best performance. In §3 we show that the blocked methods maintain the excellent backward stability of the non-blocked method. In §4 we discuss the use of the new approach within the Schur method and explain how it can be extended to the real Schur method of Higham [10]. We compare our serial implementations with existing MATLAB functions. In §5 we compare parallel implementations of the Schur method, finding that standard blocking offers the greatest speedups when the code is explicitly parallelized with OpenMP. In §6 we discuss some further applications of recursive blocking to multiplication and inversion of triangular matrices. Finally, conclusions are given in §7.

## 2 The Use of Blocking in the Schur Method

To compute $A^{1/2}$, a Schur decomposition $A = QTQ^*$ is obtained, where $T$ is upper triangular and $Q$ is unitary. Then $A^{1/2} = QT^{1/2}Q^*$. For the remainder of this section we will focus on upper triangular matrices only. The equation

$$U^2 = T \tag{1}$$

can be solved by noting that $U$ is also upper triangular, so that by equating elements,

$$U_{ii}^2 = T_{ii}, \tag{2}$$

$$U_{ii}U_{ij} + U_{ij}U_{jj} = T_{ij} - \sum_{k=i+1}^{j-1} U_{ik}U_{kj}. \tag{3}$$

These equations can be solved either a column or a superdiagonal at a time, but solving a column at a time is preferable since it allows more efficient use of cache memory. Different choices of sign in the scalar square roots of (2) lead to different matrix square roots. This method will be referred to hereafter as the "point" method.

The algorithm can be blocked by letting the $U_{ij}$ and $T_{ij}$ in (2) and (3) refer to $m \times m$ blocks, where $m \ll n$ (we assume, for simplicity, that $m$ divides $n$). The diagonal blocks $U_{ii}$ are then obtained using the point method and the off-diagonal blocks are obtained by solving the Sylvester equations (3) using LAPACK routine xTRSYL (where 'x' denotes D or Z according to whether real or complex arithmetic is used) [4]. Level 3 BLAS can be used in computing the right-hand side of (3) so significant improvements in efficiency are expected. This approach is referred to as the (standard) block method.

To test this approach, a Fortran implementation was written and compiled with gfortran on a 64 bit Intel Xeon machine, using the ACML Library for LAPACK and BLAS calls. Complex upper triangular matrices were generated, with random elements whose real and imaginary parts were chosen from the uniform distribution on $[0, 1)$. Figure 1 shows the run times for the methods, for values of $n$ up to 8000. A block size of 64 was chosen, although the speed did not appear to be particularly sensitive to the block size—similar results were obtained with blocks of size 16, 32, and 128. The block method was found to be up to 6 times faster than the point method. The residuals $\|\widehat{U}^2 - T\|/\|T\|$, where $\widehat{U}$ is the computed value of $U$, were similar for both methods. Table 1 shows that, for $n = 4000$, approximately 85% of the run time is spent in ZGEMM calls.

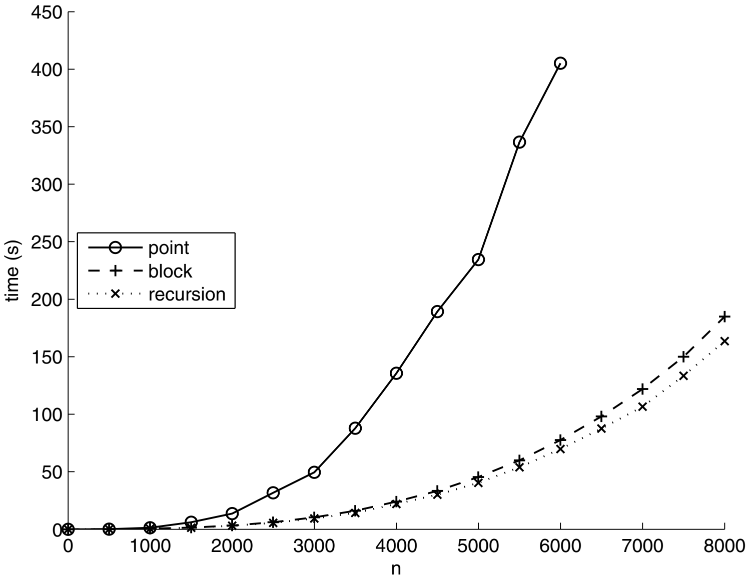

**Fig. 1.** Run times for the point, block, and recursion methods for computing the square root of a complex $n \times n$ triangular matrix for $n \in [0, 8000]$

A larger block size enables larger GEMM calls to be made. However, it leads to larger calls to the point algorithm and to xTRSYL (which only uses level 2 BLAS). A recursive approach may allow increased use of level 3 BLAS.

Equation (1) can be rewritten as

$$\begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}^2 = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}, \tag{4}$$

where the submatrices are of size $n/2$ or $(n \pm 1)/2$ depending on the parity of $n$. Then $U_{11}^2 = T_{11}$ and $U_{22}^2 = T_{22}$ can be solved recursively, until some base level is reached, at which point the point algorithm is used. The Sylvester equation $U_{11}U_{12} + U_{12}U_{22} = T_{12}$ can then be solved using a recursive algorithm devised by Jonsson and Kågström [14]. In this algorithm, the Sylvester equation $AX + XB = C$, with $A$ and $B$ triangular, is written as

$$\begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix} \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} + \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

where each submatrix is of size $n/2$ or $(n \pm 1)/2$. Then

$$A_{11}X_{11} + X_{11}B_{11} = C_{11} - A_{12}X_{21}, \tag{5}$$

$$A_{11}X_{12} + X_{12}B_{22} = C_{12} - A_{12}X_{22} - X_{11}B_{12}, \tag{6}$$

$$A_{22}X_{21} + X_{21}B_{11} = C_{21}, \tag{7}$$

$$A_{22}X_{22} + X_{22}B_{22} = C_{22} - X_{21}B_{12}. \tag{8}$$

Equation (7) is solved recursively, followed by (5) and (8), and finally (6). At the base level a routine such as xTRSYL is used.

The run times for a Fortran implementation of the recursion method in complex arithmetic, with a base level of size 64, are shown in Figure 1. The approach was found to be consistently 10% faster than the block method, and up to 8 times faster than the point method, with similar residuals in each case. The precise choice of base level made little difference to the run time.

Table 2 shows that the run time is dominated by GEMM calls and that the time spent in ZTRSYL and the point algorithm is similar to the block method. The largest GEMM call uses a submatrix of size $n/4$.

**Table 1.** Profiling of the block method for computing the square root of a triangular matrix, with $n = 4000$. Format: *time in seconds (number of calls).*

| Total time taken: | 24.03 |
|---|---|
| Calls to point algorithm: | 0.019 (63) |
| Calls to ZTRSYL | 3.47 (1953) |
| Calls to ZGEMM: | 20.54 (39711) |

## 3 Stability of the Blocked Algorithms

We use the standard model of floating point arithmetic [11, §2.2] in which the result of a floating point operation, op, on two scalars $x$ and $y$ is written as

$$fl(x \text{ op } y) = (x \text{ op } y)(1 + \delta), \qquad |\delta| \le u,$$

where $u$ is the unit roundoff. In analyzing a sequence of floating point operations it is useful to write [11, §3.4]

$$\prod_{i=1}^{n} (1 + \delta_i)^{\rho_i} = 1 + \theta_n, \qquad \rho_i = \pm 1,$$

**Table 2.** Profiling of the recursive method for computing the square root of a triangular matrix, with $n = 4000$. Format: *time in seconds (number of calls).*

| Total time taken: | 22.04 |
|---|---|
| Calls to point algorithm: | 0.002 (64) |
| Calls to ZTRSYL | 3.37 (2016) |
| Calls to ZGEMM total: | 18.64 (2604) |
| Calls to ZGEMM with $n = 1000$ | 7.40 (4) |
| Calls to ZGEMM with $n = 500$ | 5.34 (24) |
| Calls to ZGEMM with $n = 250$ | 3.16 (112) |
| Calls to ZGEMM with $n = 125$ | 1.81 (480) |
| Calls to ZGEMM with $n <= 63$ | 0.94 (1984) |

where

$$|\theta_n| \le \frac{nu}{1-nu} =: \gamma_n.$$

It is also convenient to define $\widetilde{\gamma}_n = \gamma_{cn}$ for some small integer $c$ whose precise value is unimportant. We use a hat denote a computed quantity and write $|A|$ for the matrix whose elements are the absolute values of the elements of $A$.

Björck and Hammarling [6] obtained a normwise backward error bound for the Schur method. The computed square root $\widehat{X}$ of the full matrix $A$ satisfies $\widehat{X}^2 = A + \Delta A$, where

$$\|\Delta A\|_F \le \widetilde{\gamma}_{n^3} \|\widehat{X}\|_F^2. \tag{9}$$

Higham [12, §6.2] obtained a componentwise bound for the triangular phase of the algorithm. The computed square root $\widehat{U}$ of the triangular matrix $T$ satisfies $\widehat{U}^2 = T + \Delta T$, where

$$|\Delta T| \le \widetilde{\gamma}_n |\widehat{U}|^2. \tag{10}$$

This bound implies (9). We now investigate whether the bound (10) still holds when the triangular phase of the algorithm is blocked.

Consider the Sylvester equation $AX + XB = C$ in $n \times n$ matrices with triangular $A$ and $B$. When it is solved in the standard way by the solution of $n$ triangular systems the residual of the computed $\hat{X}$ satisfies [11, §16.1]

$$|C - (A\widehat{X} + \widehat{X}B)| \le \widetilde{\gamma}_n (|A||\widehat{X}| + |\widehat{X}||B|). \tag{11}$$

In the (non-recursive) block method, to bound $\Delta T_{ij}$ we must account for the error in performing the matrix multiplications on the right-hand side of (3). Standard error analysis for matrix multiplication yields, for blocks of size $m$,

$$\left| fl\left( \sum_{k=i+1}^{j-1} \widehat{U}_{ik}\widehat{U}_{kj} \right) - \sum_{k=i+1}^{j-1} \widehat{U}_{ik}\widehat{U}_{kj} \right| \le \widetilde{\gamma}_n |\widehat{U}|_{ij}^2.$$

Substituting this into the residual for the Sylvester equation in the off-diagonal blocks, we obtain the componentwise bound (10).

To obtain a bound for the recursive blocked method we must first check if (11) holds when the Sylvester equation is solved using Jonsson and Kågström's recursive algorithm. This can be done by induction, assuming that (11) holds at the base level. For the inductive step, if suffices to incorporate the error estimates for the matrix multiplications in the right hand sides of (5)–(8) into the residual bound.

Induction can then be applied to the recursive blocked method for the square root. The bounds (10) and (11) are assumed to hold at the base level. The inductive step is similar to the analysis for the block method. Overall, (10) is obtained.

We conclude that both our blocked algorithms for computing the matrix square root satisfy backward error bounds of the same forms (9) and (10) as the point algorithm.

## 4 Serial Implementations

When used with full (non-triangular) matrices, more modest speedups are expected because of the significant overhead in computing the Schur decomposition. Figure 2 compares run times of the MATLAB function `sqrtm` (which does not use any blocking) and Fortran implementations of the the point method (`fort_point`) and the recursive blocked method (`fort_recurse`), called from within MATLAB using a mex interface, on a 64 bit Intel i3 machine. The matrices have elements whose real and imaginary parts are chosen from the uniform random distribution on the interval $[0, 1)$. The recursive routine is found to be up to 2.5 times faster than `sqrtm` and 2 times faster than `fort_point`.

An extension of the Schur method due to Higham [10] enables the square root of a real matrix to be computed without using complex arithmetic. A real Schur decomposition of $A$ is computed. Square roots of the $2 \times 2$ diagonal blocks of the upper quasi-triangular factor are computed using an explicit formula. The recurrence (3) now proceeds either a block column or a block superdiagonal at a time, where the blocks are of size $1 \times 1$, $1 \times 2$, $2 \times 1$, or $2 \times 2$ depending on the diagonal block structure. A MATLAB implementation of this algorithm `sqrtm_real` is available in the Matrix Function Toolbox [9]. The algorithm can also be implemented in a recursive manner, the only subtlety being that the "splitting point" for the recursion must be chosen to avoid splitting any $2 \times 2$ diagonal blocks. A similar error analysis to §3 applies to the real recursive method, though since only a normwise bound is available for the point algorithm applied to the quasi-triangular matrix the backward error bound (10) holds in the Frobenius norm rather than elementwise.

Figure 3 compares the run times of `sqrtm` and `sqrtm_real` with Fortran implementations of the real point method (`fort_point_real`) and the real recursive method (`fort_recurse_real`), also called from within MATLAB. The matrix elements are chosen from the uniform random distribution on $[0, 1)$. The recursive routine is found to be up to 6 times faster than `sqrtm` and `sqrtm_real` and 2 times faster than `fort_point_real`.

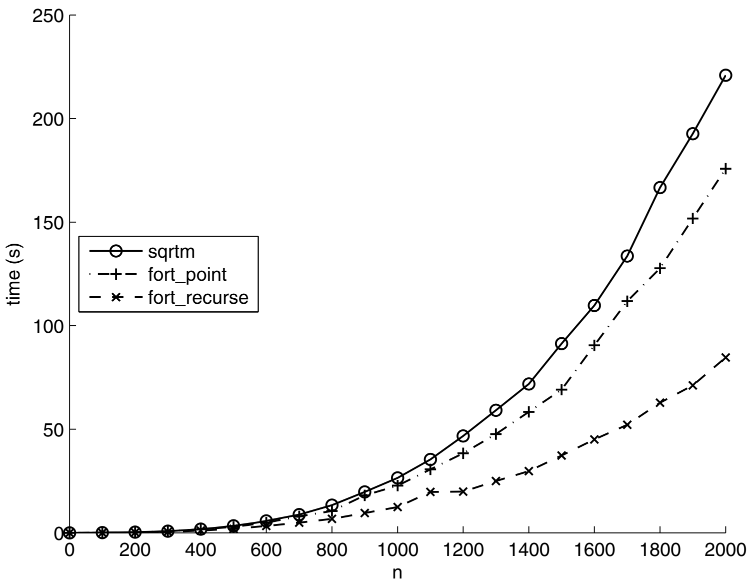

**Fig. 2.** Run times for `sqrtm`, `fort_recurse`, and `fort_point` for computing the square root of a full $n \times n$ matrix for $n \in [0, 2000]$

Both the real and complex recursive blocked routines spend over 90% of their run time in computing the Schur decomposition, compared with 44% for `fort_point`, 46% for `fort_point_real`, 25% for `sqrtm`, and 16% for `sqrtm_real`. The latter two percentages reflect the overhead of the MATLAB interpreter in executing the recurrences for the (quasi-) triangular square root phase. The 90% figure is consistent with the flop counts of $28n^3$ flops for computing the Schur decomposition and transforming back from Schur form and $n^3/3$ flops for the square root of the triangular matrix.

## 5 Parallel Implementations

The blocked and recursive algorithms allow parallel architectures to be exploited simply by using threaded BLAS. Further performance gains might be extracted by explicitly parallelizing the triangular phase using OpenMP.

In (3), the $(i, j)$ element of $U$ can be computed only after the elements to its left in the $i$th row and below it in the $j$th column have been found. Computing $U$ by column therefore offers no opportunity for parallelism within the column computation. Instead we will compute $U$ by superdiagonal, which allows the elements on each superdiagonal to be computed in parallel. Parallelization of the blocked algorithm is analogous.

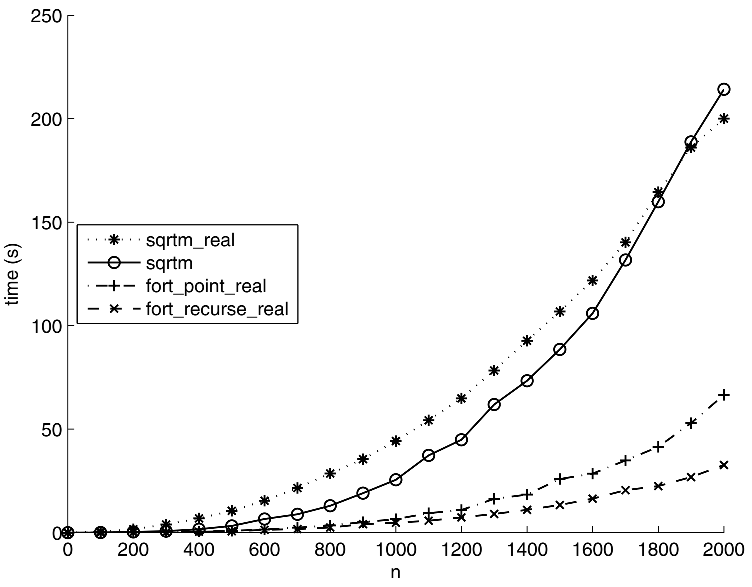

**Fig. 3.** Run times for `sqrtm`, `sqrtm_real`, `fort_recurse_real` and `fort_point_real` for computing the square root of a full $n \times n$ matrix for $n \in [0, 2000]$

The recursive block method can be parallelized using OpenMP tasks. Each recursive call generates a new task. Synchronization points are required to ensure that data dependencies are preserved. Hence, in equation (4), $U_{11}$ and $U_{22}$ can be computed in parallel, and only then can $U_{12}$ be found. When solving the Sylvester equation recursively, only (5) and (8) can be solved in parallel.

When sufficient threads are available (for example when computing the Schur decomposition) threaded BLAS should be used. When all threads are busy (for example during the triangular phase of the algorithm), serial BLAS should be used, to avoid the overhead of creating threads unnecessarily. Unfortunately, it is not possible to control the number of threads available to individual BLAS calls in this way. In the implementations described below threaded BLAS are used throughout, despite this “overparallelization” overhead.

The parallelized Fortran test codes were compiled on a machine containing 4 Intel Xeon CPUs, with 8 available threads, linking to ACML threaded BLAS [1]. Figure 4 compares run times for the triangular phase of the algorithm, with triangular test matrices generated with elements having real and imaginary parts chosen from the uniform random distribution on the interval $[0, 1)$.

The point algorithm does not use BLAS, but 2-fold speedups on eight cores are obtained using OpenMP. With standard blocking, threaded BLAS alone gives a 2-fold speed up, but using OpenMP gives a 5.5 times speedup. With recursive blocking, a 3-fold speedup is obtained by using threaded BLAS,

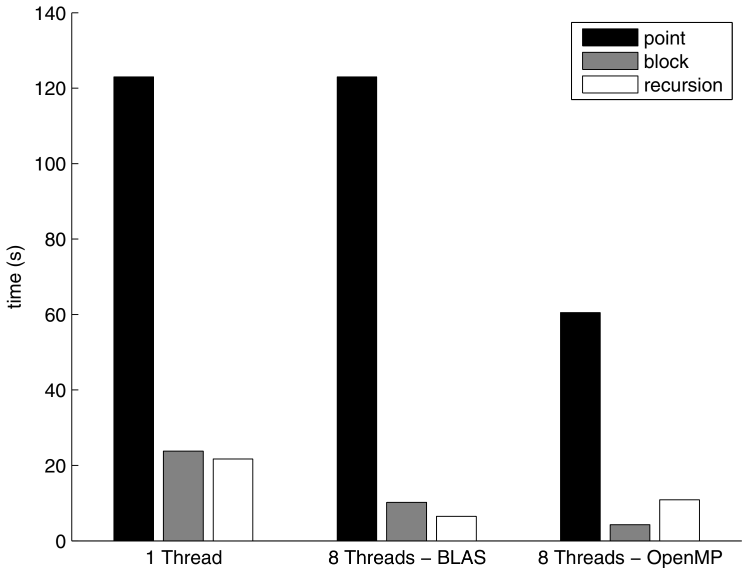

**Fig. 4.** Run times for parallel implementations of the point, block, and recursion methods for computing the square root of a $4000 \times 4000$ triangular matrix

but using OpenMP then decreases the performance because of the multiple synchronization points at each level of the recursion. Overall, if the only parallelization available is from threaded BLAS, then the recursive algorithm is the fastest. However, if OpenMP is used then shorter run times are obtained using the standard blocking method.

Figure 5 compares run times for computing the square root of a full square matrix. Here, the run times are dominated by the Schur decomposition, so the most significant gains are obtained by simply using threaded BLAS and the gains due to the new triangular algorithms are less apparent.

## 6 Further Applications of Recursive Blocking

We briefly mention two further applications of recursive blocking schemes.

Currently there are no LAPACK or BLAS routines designed specifically for multiplying two triangular matrices, $T = UV$ (the closest is the BLAS routine xTRMM which multiplies a triangular matrix by a full matrix). However, a block algorithm is easily derived by partitioning the matrices into blocks. The product of two off-diagonal blocks is computed using xGEMM. The product of an off-diagonal block and a diagonal block is computed using xTRMM. Finally the point method is used when multiplying two diagonal blocks.

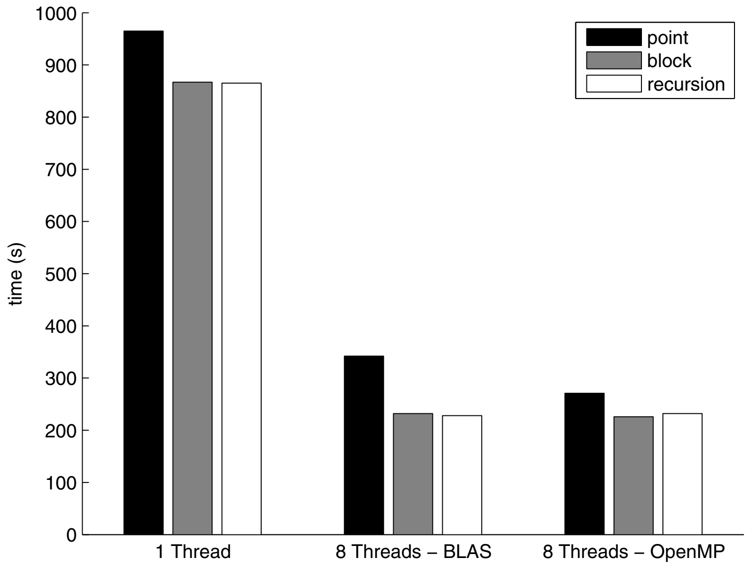

**Fig. 5.** Run times for parallel implementations of the point, block, and recursion methods for computing the square root of a 4000 × 4000 full matrix

In the recursive approach, $T = UV$ is rewritten as

$$\begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix} = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} \begin{pmatrix} V_{11} & V_{12} \\ 0 & V_{22} \end{pmatrix}.$$

Then $T_{11} = U_{11}V_{11}$ and $T_{22} = U_{22}V_{22}$ are computed recursively and $T_{12} = U_{11}V_{12} + U_{12}V_{22}$ is computed using two calls to xTRMM.

Figure 6 shows run times for some triangular matrix multiplications using serial Fortran implementations of the point method, standard blocking, and recursive blocking on a single Intel Xeon CPU (the block size and base levels were both 64 in this case, although the results were not too sensitive to the precise choice of these parameters). As for the matrix square root, the block algorithms significantly outperform the point algorithm, with the recursive approach outperforming the standard blocking approach by approximately 5%. However, if the result of the multiplication is required to overwrite one of the matrices (so that $U \leftarrow UV$, as is the case in xTRMM) then standard blocking may be preferable because less workspace is required.

The inverse of a triangular matrix can be computed recursively, by expanding $UU^{-1} = I$ as

$$\begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} \begin{pmatrix} (U^{-1})_{11} & (U^{-1})_{12} \\ 0 & (U^{-1})_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}.$$

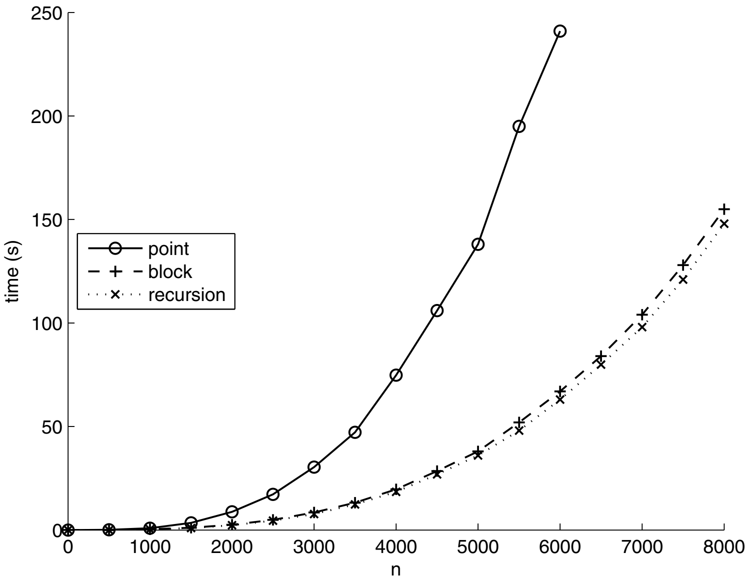

**Fig. 6.** Run times for the point, block, and recursion methods for multiplying randomly generated $n \times n$ triangular matrices for $n \in [0, 8000]$

Then $(\widehat{U}^{-1})_{11}$ and $(\widehat{U}^{-1})_{22}$ are computed and $(\widehat{U}^{-1})_{12}$ is obtained by solving $U_{11}(\widehat{U}^{-1})_{12} + U_{12}(\widehat{U}^{-1})_{22} = 0$. Provided that forward substitution is used, the right (or left) recursive inversion method can be shown inductively to satisfy the same right (or left) elementwise residual bound as the point method [7]. A Fortran implementation of this idea was found to perform similarly to LAPACK code xTRTRI, so no real benefit was derived from recursive blocking.

## 7 Conclusions

We investigated two different blocking techniques within Björck and Hammarling's recurrence for computing a square root of a triangular matrix, finding that in serial implementations recursive blocking gives the best performance. Neither approach entails any loss of backward stability. We implemented the recursive blocking with both the Schur method and the real Schur method (which works entirely in real arithmetic) and found the new codes to be significantly faster than corresponding point codes, which include the MATLAB functions `sqrtm` (built-in) and `sqrtm_real` (from [9]). Parallel implementations were investigated using a combination of threaded BLAS and explicit parallelization via OpenMP. When the only parallelization comes from threaded BLAS recursive blocking still gives the best performance. However, when OpenMP is used better performance is obtained using standard blocking. The new codes will appear in a future mark of the NAG Library [15]. Since future marks of the NAG Library will be implemented explicitly in parallel with OpenMP, the standard blocking algorithm will be used. Recursive blocking is also fruitful for multiplying triangular matrices.

Because of the importance of the (quasi-) triangular square root, which arises in algorithms for computing the matrix logarithm [2], [3], matrix $p$th roots [5], [8], and arbitrary matrix powers [13], this computational kernel is a strong contender for inclusion in any future extensions of the BLAS.

## References

1. Advanced Micro Devices, Inc., Numerical Algorithms Group Ltd. AMD Core Math Library (ACML), 4.1.0 edn. (2008)
2. Al-Mohy, A.H., Higham, N.J.: Improved inverse scaling and squaring algorithms for the matrix logarithm. SIAM J. Sci. Comput. 34(4), C153–C169 (2012)
3. Al-Mohy, A.H., Higham, N.J., Relton, S.D.: Computing the Fréchet derivative of the matrix logarithm and estimating the condition number. MIMS EPrint 2012.72. Manchester Institute for Mathematical Sciences. The University of Manchester, UK (2012)
4. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. Society for Industrial and Applied Mathematics. Philadelphia, PA (1999)
5. Bini, D.A., Higham, N.J., Meini, B.: Algorithms for the matrix $p$th root. Numer. Algorithms 39(4), 349–378 (2005)
6. Björck, Å., Hammarling, S.: A Schur method for the square root of a matrix. Linear Algebra Appl. 52/53, 127–140 (1983)
7. Du Croz, J.J., Higham, N.J.: Stability methods for matrix inversion. IMA J. Numer. Anal. 12(1), 1–19 (1992)
8. Guo, C.-H., Higham, N.J.: A Schur–Newton method for the matrix $p$th root and its inverse. SIAM J. Matrix Anal. Appl. 28(3), 788–804 (2006)
9. Higham, N.J.: The Matrix Function Toolbox, `http://www.ma.man.ac.uk/~higham/mftoolbox`
10. Higham, N.J.: Computing real square roots of a real matrix. Linear Algebra Appl. 88/89, 405–430 (1987)
11. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM (2002)
12. Higham, N.J.: Functions of Matrices: Theory and Computation. SIAM (2008)
13. Higham, N.J., Lin, L.: A Schur–Padé algorithm for fractional powers of a matrix. SIAM J. Matrix Anal. Appl. 32(3), 1056–1078 (2011)
14. Jonsson, I., Kågström, B.: Recursive blocked algorithms for solving triangular systems - part I: One-sided and coupled Sylvester-type matrix equations. ACM Trans. Math. Software 28(4), 392–415 (2002)
15. Numerical Algorithms Group. The NAG Fortran Library, `http://www.nag.co.uk`