References

[AG] Ammar, G. S., and W. G. Gragg, "Superfast Solution of Real Positive Definite Toeplitz Systems," *SIAM J. on Matrix Analysis and Applications*, vol. 9, 1, pp. 61-76, 1988.

[BG] Bini, D., Gemigniani, L., "On the Euclidean Scheme for Polynomials Having Interlaced Real Zeros", manuscript, 1990.

[BA] Bitmead, R. R., and B. D. O. Anderson, "Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations," *Linear Algebra and its Applics.*, vol. 34, pp. 103-116, 1980.

[Br] Brent, R.P., "The Parallel Evaluation of General Arithmetic Expressions", *J. ACM,* **21**, pp. 201-208, 1974.

[CKL-A] Chun, J., T. Kailath, and H. Lev-Ari, "Fast Parallel Algorithm for QR-factorization of Structured Matrices," *SIAM J. on Scientific and Statistical Computing*, vol. 8,6, pp. 899-913, 1987.

[CB] Cybenko, G., and M. Berry, "Hyperbolic Householder Algorithm for Factoring Structured Matrices", to appear in *SIMAX.*

[GKK] Gohberg, I., T. Kailath, and I. Koltracht, "Efficient Solution of Linear Systems of Equations with Recursive Structure," *Linear Algebra and Its Applics.*, vol. 80, pp. 81-113, 1986.

[GL] Golub, G. H., and C. F. van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, Maryland, 1989.

[KKM] Kailath, T., S.-Y. Kung, and M. Morf, "Displacement Ranks of Matrices and Linear Equations," *J. Math. Anal. Appl.*, vol. 68,2, pp. 395-407, 1979.

[KR] Karp, R. M., and V. Ramachandran, "A Survey of Parallel Algorithms for Shared Memory Machines," Report No. UCB/CSD 88/408, Computer Science Division, University of California, Berkeley, CA, 1988 (to appear in *Handbook of Theoretical Computer Science*, North-Holland, Amsterdam).

[M] Muscius, B. R., "Levinson and Fast Choleski Algorithms for Toeplitz and Almost Toeplitz Matrices", Internal Report, *Lab. of Electronics, M.I.T.*, 1981.

[P89] Pan, V., "Fast and Efficient Parallel Inversion of Toeplitz and Block Toeplitz Matrices," *Operator Theory; Advances and Applications,* vol. 40, pp. 359-389, 1989.

[P89a] Pan, V., "On Some Computations with Dense Structured Matrices," *Proc. ACM-SIGSAM Symposium ISSAC-89*, pp. 34-42, 1989 (to appear in *Math. of Computations*).

[PR89] Pan, V., and J. Reif, "Fast and Efficient Solution of Path Algebra Problems," *J. Comp. and System Sciences*, vol. 38, 3, pp. 494-510, 1989.

(and therefore, the complexity of computing LU- and LDM^T-factorizations) certainly within the bounds (4.4). This also implies the same complexity bounds for the extended Euclidean computations of [BG].

communication and synchronization where $p_I(h)$ is large, but such computations are theoretically important; they may also have some practical impact where the matrices are structured and diagonally dominant, see [P89]).

We may assume that $p_I(h)$ exceeds $\hat{c}h$ for some positive constant $\hat{c}$, set $s_i = 1$ for all i and deduce from (4.1)-(4.3), that

$$t_F(n) < \frac{c}{d+1}(\log_2 n + 1)^{d+1} \ , \quad p_F(n) = p_I(n) \ ,$$

but if, say, $p_I(h) \geq \hat{c}h^{\hat{d}}$ for two constants $\hat{c} > 0$ and $\hat{d} > 1$, then we may also set $s_i = 2^i$, for $i = 0, \ldots, i_0$, $i_0 = \lceil \frac{1}{\hat{d}-1} \log_2 \log_2 n \rceil$; set $s_i = 1$, for $i = i_0 + 1, \ldots, \log_2 n$, and deduce that

$$\begin{aligned} t_F(n) &< \frac{c}{d+1}(\log_2 n - i_0 + 1)^{d+1} + \frac{c(\log_2 n)^{d+1}}{1-(1/2)^{\hat{d}-1}} \\ &< \big(\frac{1}{d+1} + \frac{1}{1-(1/2)^{\hat{d}-1}}\big) c(\log_2 n)^{d+1} \ , \\ p_F(n) &= \frac{p_I(n)}{\log_2 n} \ , \end{aligned}$$

thus saving processors for the cost of a rather minor increase of the time bound: if $d = 2$ the above bound gives us that

$$t_F(n) < (\frac{1}{d+1} + 2)c(\log_2 n)^{d+1} \ , \quad p_F(n) = \frac{p_I(n)}{\log_2 n} \ .$$

The latter bounds turn into

$$t_F(n) = O(\log^3 n) \ , \quad p_F(n) = \lceil n^2 / \log n \rceil \ , \tag{4.4}$$

(compare the introduction) if we assume that $d = 2$, $p_I(n) = n^2$, which is supported by the algorithms of [P89], [P90] where the entries of A are rational and can be immediately extended to Hankel and Hankel-like matrices.

Finally, we note that the conversion of the recursive factorization of an $n \times n$ Toeplitz or Hankel matrix into LU- or LDM^T-factorizations is reduced to FFTs [by using the representations of the forms (3.5), (3.6)], which keeps the complexity of such a conversion

4. Parallel complexity estimates: Let us estimate the parallel computational complexity of the recursive factorization of A and A^{-1}, assuming the customary arithmetic PRAM model of computation, where in each step each processor performs at most one arithmetic operation. Furthermore, we will follow [Br] and [KR] and will assume that the computation using time t and p processors can be slowed down and performed in time ts by using $\lceil p/s \rceil$ processors for any positive integer s.

Let hereafter, $t_I(h)$ and $t_F(h)$ denote the parallel time bounds for the inversion and factorization of an $h \times h$ matrix, whereas $p_I(h)$ and $p_F(h)$ denote the associated processor bounds. Then we immediately deduce from (2.1) and (2.3) that

$$t_F(n) \le \sum_{i=1}^{k} s_i t_I(\frac{n}{2^i}) \ , \tag{4.1}$$

$$p_F(n) \le \max_{0 \le i \le k} \left(\lceil \frac{1}{s_i} p_I(\frac{n}{2^i}) \rceil 2^{i+1} \right) \tag{4.2}$$

where $k = \log_2 n$ and, for every i, a positive integer s_i denotes the slowdown factor that we select at stage i of the recursive factorization.

In particular, we arrive at the estimates

$$t_I(h) = ch \ , \quad p_I(h) = c^* h^2$$

with two positive constants c and c^* if we apply Gaussian elimination to invert a general $h \times h$ matrix. Then we may just let $s_i = 1$ for all i and obtain that

$$t_F(n) \le 2cn \ , \quad p_F(n) \le 2c^* n^2 \ .$$

Let us also show the estimates for the similar NC computations, where we assume that

$$t_I(h) \le c(\log_2 h)^d \ , \quad p_I(h) \le c^* h^{d^*} \tag{4.3}$$

for some positive constants c, d, c^* and d^*. (NC computations for matrix inversion and factorization do not generally seem to be practical, due to the high cost of processor

This bound can be further decreased since the entires of a dg of A can also be used in the dgs of B_0 and C_0, so that the dgs of B_0 and C_0 need at most n new entries. Likewise, a dg of A_1^{-1} requires at most n entries that are not in the dg of A^{-1}, and similarly at the next steps of the recursive factorization. These observations lead us to the bound

$$S_F(n) \le 4rn + (2r+1)n \log_2 n \ . \tag{3.9}$$

To estimate the time bound $T_F(n)$ in the case of matrices A given with their dgs of length r, assume the known bound

$$T_I(h) = O(hr^2 \log^2 h)$$

(see [BA], [M], [AG]), which turns into

$$T_I(h) \le 8h \log^2 h$$

for Toeplitz matrices (see [AG]), and then we arrive at the bounds

$$T_F(n) = O(nr^2 \log^3 n) \ ,$$

$$T_F(n) \le 9n + 16n \sum_{i=1}^{\log_2 n - 1} i^2 \le (16/3)n(-0.5 + \log_2 n)^3 + 9n \ ,$$

respectively.

Both of these bounds improve the time bound of (3.3) (assuming that r is small). Furthermore, for comparison, LU-factorization of an $n \times n$ structured matrix of this class requires at least n^2 words of memory and n^2 arithmetic operations.

For Toeplitz matrices of smaller size, the user frequently prefers the algorithms supporting the bound

$$T_I(h) = O(h^2)$$

(see [CB]), but even in this case, the upper estimates of (3.8) and (3.9) are much less than the space bound n^2 for the LU-factorization.

is called a *displacement generator* (hereafter, we abbreviate this as *dg*) of length r for A. Then $GH^T = F(A)$, the image of A in its displacement in the diagonal direction either downward [assuming (3.5)] or upward [assuming (3.6)].

We refer the reader to [KKM], [CKL-A], [GKK] and [P89a] for the introduction and the study of this highly important class of matrices and of its extensions, including, in particular, the matrices represented similarly to (3.5) and (3.6), but with Hankel and other structured matrices replacing Toeplitz matrices. As an exercise, the reader may work out the extension of our complexity estimates below to the case of such structured matrices.

Hereafter, let *the Toeplitz-like matrices A of* (3.5) *and* (3.6) *be represented by their dgs*, and then the storage space for an $m \times n$ matrix A of (3.5) or (3.6) is $2rn$, which is smaller than n^2 if $n > 2r$; moreover, *we will identify the inversion* of A *with computing a dg* of A^{-1}, and similarly for the factorization of A. (3.5) and/or (3.6) for a nonsingular matrix A imply the existence of the similar dual representations [(3.6) and/or (3.5), respectively] with the same r for A^{-1} ([KKM]), and thus, we will change the space bound of (3.3) into

$$S_I(h) = O(2hr) \ , \tag{3.7}$$

for the $h \times h$ matrices given with their dgs of length r. Then (3.2) and (3.7) imply that $s_F(n) < 3r^* n \log_2 n$, r^* being an upper bound on the lengths of the dgs computed for all the matrices involved in the recursive factorization. A_0, B_0, C_0 and D_0 have dgs of lengths $r+1$ or less [examine the displaced images $F(A_0)$, $F(B_0)$, $F(C_0)$ and $F(D_0)$], and so have their inverses, for which we shall use their dual displacement representation and dual dgs. Based on such observations, we may show that, under (3.5),

$$r^* \le r + 1 \ ,$$

which implies that

$$S_F(n) < 3(r+1)n \log_2 n \ . \tag{3.8}$$

3. Sequential complexity of recursive factorization: Next, we will estimate the complexity $T_F(n)$ and $S_F(n)$ of the evaluation and of the storage of the recursive factorization of section 2 (defined for both nonsingular $n \times n$ matrix A and its inverse A^{-1}) provided that $T_I(h)$ arithmetic operations and $S_I(h)$ words of memory suffice to evaluate and to store the inverse of every $h \times h$ auxiliary matrix of this factorization.

We deduce from (2.1)-(2.3) that

$$T_F(n) \le \sum_{i=1}^{k} (2^{i-1} T_I(n/2^{i-1}) + 2^i T_I(n/2^i)) = T_I(n) + nT_I(1) + \sum_{i=1}^{k-1} 2^{i+1} T(n/2^i) \ , \quad (3.1)$$

$$S_F(n) \le 3 \sum_{i=1}^{k} 2^{i-1} S_I(n/2^i) \ . \quad (3.2)$$

For the inversion of a general $h \times h$ matrix, we have

$$T_I(h) \ge ch^2 \ , \quad S_I(h) \ge ch^2 \quad \text{for a positive constant } c \ , \quad (3.3)$$

and (3.1)-(3.3) together imply that

$$T_F(n) < 3T_I(n) \ , \quad S_F(n) < (3/2) S_I(n) \ . \quad (3.4)$$

Surely, we may decrease the storage space bounds (by roughly 50%) and the time bounds if A is symmetric.

Next, let A be a special matrix with Toeplitz-like structure represented either as

$$A = \sum_{i=1}^{r} L_i U_i \ , \quad (3.5)$$

or in the dual version (with matrices L_i and U_i and integer r generally distinct from ones of (3.5)),

$$A = \sum_{i=1}^{r} U_i L_i \ , \quad (3.6)$$

where L_i and U_i are $n \times n$ lower and upper triangular Toeplitz matrices, respectively, L_i defined by its first column g_i, and U_i by its first row h_i^T, $i = 1, \ldots, r$, and r is small relative to n. The pair of $n \times r$ matrices

$$G = [g_1, \ldots, g_r] \ , \quad H = [h_1, \ldots, h_r]$$

In the important case where A is symmetric and positive definite, all the auxiliary matrices that we need to invert (that is, the northwestern blocks, such as A_0, and their Schur complements, such as A_1, as well as their inverses, of course) are symmetric and positive definite too (so that they are nonsingular), and moreover, in this case,

$$\|A_g\|_2 \leq \|A\|_2 \ , \quad \|A_g^{-1}\|_2 \leq \|A^{-1}\|_2 \quad \text{for } g = 0, 1 \ ,$$

which follows from the interlacing property of the eigenvalues of symmetric matrices (see [GL], p. 411, or [Par], p. 186).

Finally, we note that recursively multiplying together the auxiliary matrices of the recursive factorization, we may convert it into LU- or LDM^T-factorization with lower triangular matrices L, M and U^T and a diagonal matrix D, where $L = M = U^T$ if A is symmetric.

2. Recursive triangular and orthogonal factorization: Let n be a power of 2, $n = 2^k$, A be an $n \times n$ matrix, $A = \begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix}$, and A_0, B_0, C_0 and D_0 be the $(n/2) \times (n/2)$ blocks of A. Then

$$A = \begin{bmatrix} I & O \\ C_0 A_0^{-1} & I \end{bmatrix} \begin{bmatrix} A_0 & O \\ O & A_1 \end{bmatrix} \begin{bmatrix} I & A_0^{-1} B_0 \\ O & I \end{bmatrix}, \tag{2.1}$$

$$A^{-1} = \begin{bmatrix} I & -A_0^{-1} B_0 \\ O & I \end{bmatrix} \begin{bmatrix} A_0^{-1} & O \\ O & A_1^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -C_0 A_0^{-1} & I \end{bmatrix}, \tag{2.2}$$

$$A_1 = D_0 - C_0 A_0^{-1} B_0 \ , \tag{2.3}$$

provided that A, A_0 and A_1 are nonsingular. (The matrix A_1 of the equation (2.3) is called the Schur complement of A_0.) Under the similar nonsingularity assumptions about the leading principal submatrices and their Schur complements in A_0 and A_1, we extend the decompositions (2.1)-(2.3) to the similar decompositions for the matrices A_0 and A_1. In $k = \log_2 n$ recursive steps of this process, we will arrive at 1×1 matrices and thus will complete recursive triangular factorization of the matrices A and A^{-1}. Note that $B_0 = C_0^T$ and that the matrices A_0 and A_1 are symmetric if A is symmetric.

It is customary to apply the relation (2.2)-(2.3) in order to recursively evaluate A^{-1} (see, in particular, [BA]), but we will assume that we are given an algorithm that inverts matrices, and we will compute the recursive factorization (2.2)-(2.3) given A, that is, we will recursively compute the submatrix A_0, its Schur complement A_1 in A, as well as A_0^{-1}, A_1^{-1}, and so on.

In this computation, we will apply the following simple observation (compare [BA]):

Proposition 2.1. *The matrix A_1^{-1} of the matrix equation (2.2) forms the southeastern block of the matrix A^{-1}.*

Proposition 2.1 enables us to compute the decompositions (2.1)-(2.2) solely by means of matrix inversion, that is, we will first invert the matrices A_0 and A (in particular, this will give us A_1^{-1}) and then A_1^{-1} (this will give us A_1), keeping the products $A_0^{-1} B_0$ and $C_0 A_0^{-1}$ in the factorized form, and we will similarly perform all the subsequent steps of the recursive factorization.

1. Introduction: Computing QR-factorization of an $m \times n$ matrix B of full rank for $m \geq n$ can be reduced to the Choleski factorization of the $n \times n$ symmetric and positive definite matrix $A = B^T B$, for $A = R^T R$ if $B = QR$, $Q^T Q = I$, $Q \in \mathbb{R}_{m,n}$. This approach is not popular for general matrices B (since cond A may be as large as $(\text{cond } B)^2$), but it has substantial advantages where B is a special matrix with a Toeplitz-like or a Hankel-like structure, for such a structure little deteriorates in the transition to A ([CKL-A]). Then computing (and storage) of the Choleski factorization $A = R^T R$ requires the order of n^2 arithmetic operations (and of n^2 words of memory) versus n^3 arithmetic operations in the known alternate approaches. In this paper, however, we will consider a distinct simple approach to factorization of A (and thus of B), which only requires an order of $n \log^3 n$ arithmetic operations and $O(n \log n)$ words of memory. The same approach allows a parallel implementation that uses $O(\log^3 n)$ parallel arithmetic steps and $O(n^2 / \log n)$ processors under the PRAM model of parallel computations (provided that all the entries of the input matrix are rational), and this also became the record upper estimates for the parallel complexity of computing the remainder sequence of two polynomials of degrees at most n, with rational coefficients, since the latter problem has been reduced to computing Choleski's factorization of an $n \times n$ Hankel matrix ([BG]).

In the next section we will present, or actually, revisit, this approach, which has been in use for a while in a little distinct form. In sections 3 and 4 we will estimate the sequential and parallel complexity of the resulting algorithms, respectively.

We will apply our approach to the factorization of a general (not necessarily symmetric) matrix A, which is used in some applications ([PR89]).

On a Recursive Triangular Factorization of Matrices

V. Pan

Department of Computer Science
Columbia University
New York, NY 10027
and
Department of Mathematics and Computer Science
Lehman College
CUNY, Bronx, NY 10468

and

Albany, NY 12222
(Supported by NSF Grant CCR-8805782 and by
PSC-CUNY Awards #661340, #668541 and #669290)

CUCS-026-90

Summary: We estimate the parallel and sequential complexity of computing a recursive triangular factorization of an $n \times n$ matrix, which may, in particular, replace Choleski factorization as a means of computing the QR-factorization of a matrix and may also have some other applications. The estimates for the storage space, sequential time and the number of processors required for the computation and storage of such a recursive factorization is by almost the factor of n less than the similar estimates for the Choleski factorization in the case of Toeplitz-like and Hankel-like matrices.

Key words: factorization, Toeplitz and Hankel matrices, polynomial remainder sequence.