PaA2 Deliverable
APPARC ESPRIT Contract

A Strassen-Type Matrix Inversion Algorithm

DRAFT

Susanne M. Balle
Per Christian Hansen
UNI•C
Danish Computing Center for Research and Education
Building 305, Technical University of Denmark
DK-2800 Lyngby, Denmark
Susanne.M.Balle@uni-c.dk
Per.Christian.Hansen@uni-c.dk

Nicholas J. Higham
Department of Mathematics
University of Manchester
Manchester M13 9PL
United Kingdom
higham@vtx.ma.man.ac.uk

August 10, 1993—last update by PCH

Contents

1 Introduction

Linear algebra routines play an essential role as the fundamental "building blocks" for almost all mathematical software. Hence, it is crucial to have fast and reliable implementations of linear algebra routines available on any given computer. The LAPACK Library [1] provides such software for a variety of high-performance computers, including vector computers and shared memory parallel computers. This library is based on block algorithms [11] that make extensive use of the Level 2 and 3 BLAS routines [14, Appendix D] which can be implemented very efficiently on these types of computers.

However, the software in the LAPACK Library may not be so useful on certain other types of parallel computers, including massively parallel computers such as the Connection Machines CM-200 and CM-5 and the MasPar MP-1. At first, this seems odd because the inherent data-parallelism in the block steps of the LAPACK routines seems very well suited for massively parallel computers; i.e., on these machines one would expect that good data-parallelism can be achieved on the fine-grained level (and not at the coarse-grained block level). However, data-layout and communication play an extremely important role on these machines, and it turns out to be these two issues that are most important for the speed of any algorithm. For example, the layout that may be most suited for matrix multiplication may be a very bad layout for back substitution, say, and intermediate data shuffling is very time consuming. The same holds for communication—a layout suited for matrix addition may be very bad for matrix multiplication, for example.

Hence, even though the BLAS routines—or at least some of them—can be implemented quite efficiently for specific data layouts, it is not practical to build numerical linear algebra routines on top of them. As a consequence, all the linear algebra routines in the CMSSL Library for CM-200 and CM-5 are "hand-tuned" implementations drawing upon specialized low-level routines not available to the user. For example, solution of linear systems of equations is done by a partitioned block algorithm (in the terminology of [13]) with a block-cyclic data layout; see [18] for more details. The algorithm on the MP-1 is a point algorithm with a cyclic layout.

Although such specialized software often provides very efficient implementations, it is difficult to maintain the software and also difficult to port it to new hardware and software releases and related architectures. Hence—although solution of linear systems of equations is one of the oldest topics in numerical analysis—there is still a need for developing new algorithms that are well-suited to today's high-performance computers.

1.1 Why Consider Strassen's Algorithm?

When searching for new algorithms, one may sometimes have to break with the traditional guidelines in numerical analysis and find new approaches. One approach, suggested in [10] and [5], is based on the following three-step paradigm:

1. Use a highly parallel algorithm that may sometimes fail.
2. Use a simple and reliable technique for detecting when the algorithm fails.
3. If an unreliable solution is detected, resort to some other strategy.

Regarding Step 3, Demmel [10] suggested recomputing the desired result using a slower but more reliable algorithm. This approach is useful, e.g., if the algorithm under consideration is very rarely unstable or very fast compared to the stable algorithm. As a second approach, in other situations it is possible to re-use the quantities already computed in Step 1, combined with an appropriate low-cost stabilization technique, and thus avoid a complete restart. Such a technique was suggested by Balle and Hansen in [5] in connection with matrix inversion.

The present paper extends the analysis from [5]. The key idea is that if we want to solve the linear system of equations $Ax = b$, and if an approximate inverse $\tilde{A}^{-1}$ can be computed quickly, then it may be more efficient to use this approximate inverse to compute $x = A^{-1}b$—e.g., by means of iterative refinement or by some iterative methods using $\tilde{A}^{-1}$ as a preconditioner—than to compute x by means of (block) LU factorization.

We use an asymptotically fast method for computing $\tilde{A}^{-1}$ that dates back to Strassen's 1969 paper [20] on fast matrix-matrix multiplication by a recursive algorithm. Strassen's matrix inversion algorithm is also recursive and its complexity is $\mathcal{O}(n^{\log_2 7})$ operations when the recursion is continued down to the 1×1 level, where n is the order of A. In practice one does not normally recur down to the 1×1 level, since a lower operation count and lower computational overheads are obtained by terminating the recursion early. An important feature of Strassen's inversion algorithm is that it is based on operations on block matrices, and these may provide enough data parallelism to make the algorithm suited for massively parallel computers. We remark that iterative improvement of $\tilde{x} = \tilde{A}b$, once $\tilde{A}^{-1}$ has been computed, is only an $\mathcal{O}(n^2)$ process.

Our use of Strassen's matrix inversion algorithm is motivated by the success of Strassen's matrix multiplication method [20], which has turned out to be useful on high-performance computers [2, 4, 9, 16]. It is found that although the rounding error properties of this algorithm are less favorable than for conventional matrix multiplication, the algorithm is still useful as a Level 3 BLAS routine in many circumstances.

As we shall see in §3.1, Strassen's matrix inversion algorithm is fundamentally unstable. The main reason is that it involves the inversion of submatrices on each recursion level. The key idea in the algorithm we develop is that if one of these submatrices is ill-conditioned, then the computation is stabilized by perturbing this submatrix to make it better conditioned. As long as the perturbation is small enough, the computed approximate inverse $\tilde{A}^{-1}$ can still be used for refining $\tilde{x} = \tilde{A}^{-1}b$ as outlined above. It is this stabilization technique, in which the data are re-used instead of using a different algorithm, that puts this algorithm in the second class of algorithms falling under the three-step paradigm outlined above.

A crucial point in the above algorithm is the decision when to add stabilization to a submatrix, and how much stabilization to add. There is obviously a tradeoff between adding too much stabilization, so that the approximate inverse $\tilde{A}^{-1}$ is too far from the exact A^{-1}, and too little stabilization, so that the computed $\tilde{A}^{-1}$ is too much contaminated by the effects of rounding errors (which are proportional to the condition number of the particular ill-conditioned submatrix or submatrices). One important goal of our analysis here is to guide this decision.

It this paper, we focus on the computation of the approximate inverse $\tilde{A}^{-1}$, since this is the crucial step in solving linear systems $Ax = b$ via Strassen's method. We make the important assumption that the matrix A is well-conditioned; otherwise, other methods (based on, e.g., regularization) should usually be used anyway.

1.2 Some Related Issues

We want to mention at this point that our interest in this algorithm is also motivated by one of the so-called "superfast" algorithms for inversion of a Toeplitz matrix, described by Bitmead & Anderson [8]. This algorithm is mathematically equivalent to Strassen's inversion algorithm; however, the operations are performed by means of FFTs, using the low displacement rank structure of the involved submatrices. In this way, and $\mathcal{O}(n \log^2 n)$ algorithm is obtained. Since the Bitmead-Anderson algorithm is based on

Strassen's inversion algorithm, our stabilization technique can easily be applied to the Bitmead-Anderson algorithm. We shall not pursue this aspect further here, but leave it to a separate investigation.

Two quite different methods for stabilizing Strassen's matrix inversion algorithm are proposed by Bailey & Ferguson in [3]. Their first method is based on the observation that if one submatrix is ill-conditioned, then one can instead invert another submatrix which cannot be ill conditioned as long as A is well conditioned. However, it is difficult to quantify when this form of "pivoting" should be used, and there is no way to improve the accuracy of the method for submatrices with medium-size condition numbers. On the other hand, our stabilization technique can be applied to any submatrix, even (in principle) well-conditioned ones.

Bailey & Ferguson also suggest the use of Newton iteration to improve the accuracy of the computed inverses of the sub-blocks at each level of recursion and, optionally, of the overall computed inverse. At this point, it is not clear which approach is more efficient: to refine $\tilde{A}^{-1}$ before computing the solution, or to compute $\tilde{x} = \tilde{A}^{-1}b$ and refine this vector.

1.3 Organization of the Report

The key new results in the present paper, compared to [5], are the following:

1. A refined forward error analysis.
2. A better understanding of how to choose the amount of stabilization.
3. Results from implementing the algorithm on a CM-200.

Our paper is organized as follows. In Section 2 we summarize Strassen's matrix inversion algorithm and discuss some issues related to data layout. In Section 3 we apply standard forward error analysis to the algorithm, and we compare with numerical tests. Next, in Section 4 we propose a method for stabilization of Strassen's algorithm, and in Section 5 we present some results obtained on a CM-200. Finally, in Section 6 we mention some open problems and we point to future research.

2 Strassen's Matrix Inversion Algorithm and Data Layout

In this section we summarize Strassen's asymptotically fast algorithm for matrix inversion, and we discuss some issues related to data layout.

2.1 The Algorithm

Throughout, for convenience we assume that the matrix A is $n \times n$ where n is a power of 2, and that we split A and its inverse into four square submatrices of order $n/2$. Strassen's algorithm is based on the following well-known formula for the inverse of

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}. \tag{2.1}$$

Then the inverse $C = A^{-1}$ is given by

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{pmatrix}, \tag{2.2}$$

where S is the Schur complement of A_{11}, defined as

$$S \equiv A_{22} - A_{21}A_{11}^{-1}A_{12}. \tag{2.3}$$

The evaluation of C via (2.2) and (2.3) can be expressed as follows, using intermediate matrices $R_1, \ldots, R_6$:

ALGORITHM MATINV

$$\begin{array}{rrcl} 1. & R_1 & = & A_{11}^{-1} \\ 2. & R_2 & = & A_{21}R_1 \\ 3. & R_3 & = & R_1A_{12} \\ 4. & R_4 & = & A_{21}R_3 \\ 5. & S & = & A_{22} - R_4 \\ 6. & R_5 & = & S^{-1} \\ 7. & C_{22} & = & R_5 \\ 8. & C_{12} & = & -R_3R_5 \\ 9. & C_{21} & = & -R_5R_2 \\ 10. & R_6 & = & C_{12}R_2 \\ 11. & C_{11} & = & R_1 - R_6 \end{array}$$

This algorithm is identical to Strassen's original algorithm when the following techniques are employed:

$$\begin{array}{c|c} \text{Node 1,1} & \text{Node 1,2} \\ \hline \text{Node 2,1} & \text{Node 2,2} \end{array} = \begin{array}{cc|cc} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ \hline 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{array} = \begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}$$

Figure 2.1: Default layout of a 4×4 matrix on a 2×2 nodal array.

- ALGORITHM MATINV is used recursively in Steps 1 and 6 to compute the inverses of all the submatrices, and recursion is continued down to the 1×1 level.
- Strassen's matrix-matrix multiplication algorithm is used to perform all the matrix multiplications (Steps 2, 3, 4, 8, 9 and 10).

We shall not give the details for Strassen's matrix-matrix multiplication algorithm (they can be found in many references, e.g., [20], [16]), but we recall that this algorithm requires $n^{\log_2 7} \approx n^{2.807}$ multiplications (when full recursion is used). Using this fact it is easy to show that the number of multiplications used in Strassen's inversion algorithm is

$$\frac{6}{5} n^{\log_2 7} - \frac{1}{5} n \approx \frac{6}{5} n^{2.807}. \tag{2.4}$$

In practical implementations, because of the overhead associated with subroutine calls, etc., one does not use full recursion, but instead one switches to some other algorithm at an appropriate block size; see [16]. In the matrix inversion algorithm, we suggest switching to computing the inverse via LU factorization at the lowest level.

2.2 Data Layout

For massively parallel computers such as the CM-200, CM-5 and MP-1, it is important to take into account the layout of the data elements on the processing nodes. Often, the default layout is not the most appropriate one. These issues are discussed in [10] and [6].

When the ALGORITHM MATINV is implemented with Strassen's matrix-matrix multiplication method it requires a large number of matrix additions. These operations can be executed quickly provided that the data are allocated to the processors in the right way. We illustrate this with a simple example using a 4×4 matrix. If the elements of this matrix are distributed

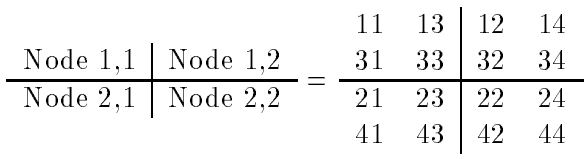

Figure 2.2: Alternative layout of a 4×4 matrix on a 2×2 nodal array. Now, the matrix elements residing on one processor node do not correspond to a block of A.

on a 2×2 processor grid using the typical default layout, shown in Figure 2.1, then adding any two different submatrices A_{ij} causes communication. However, if the elements are distributed to the processors as shown in Figure 2.2 then no communication is required to add any two submatrices. Other types of additions may require other layouts, including the default layout; but for the submatrix additions used in ALGORITHM MATINV the layout in Figure 2.2 is optimal. On the CM-200, the desired layout is termed `(SERIAL,SERIAL,NEWS,NEWS)`.

3 Forward Error Analysis

In this section we present a forward error analysis of ALGORITHM MATINV from the previous section. Because of the recursive nature of the algorithm, forward error analysis seems to be the only way to analyze this algorithm, and yet still—as we shall see below—there are some open problems.

3.1 The Inherent Instability of the Method

Before carrying out the analysis, it is important to realize that the method under consideration is inherently unstable.

To see this, note that the formulae underlying Strassen's method can be obtained by forming the block LU factorization

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ 0 & S \end{pmatrix},$$

from which we have

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{pmatrix}.$$

Strassen's formulae are obtained on multiplying out the product. This gives the first insight into why Strassen's method is unstable: the block LU factorization on which it is based is not backward stable in general [13], so we would expect Strassen's method also to be unstable, even if only one level of recursion and conventional multiplication is used.

Next, suppose A is upper triangular. Then Strassen's formulae reduce to

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} & -A_{11}^{-1}A_{12}A_{22}^{-1} \\ 0 & A_{22}^{-1} \end{pmatrix}.$$

This is essentially Method 2B in [15], which is found not to give a small residual. The cure used in [15] is to convert one of the multiplies into a multiple right-hand side triangular solve, so that we form, say $A_{11}^{-1}A_{12}/A_{22}$ (where / denotes back substitution). However, this changes Strassen's method and there does not seem to be an analogue of this modification for the full matrix case.

Finally, we note that Strassen's inversion method is not guaranteed to be stable even for symmetric positive definite matrices, as numerical examples in [17] show.

3.2 Theoretical Error Bounds

We use the standard notation in which $fl(\cdot)$ denotes the *computed* version of the particular quantity. Moreover, we let E_i, F_i and G_i denote the error matrices corresponding to matrix inversion, matrix multiplication, and matrix addition, respectively, in the various steps of the algorithm. Thus, for ALGORITHM MATINV we obtain the following results.

$$fl(A_{11}^{-1}) = A_{11}^{-1} + E_1$$

$$fl(S) = A_{22} - [A_{21}((A_{11}^{-1} + E_1)A_{12} + F_1) + F_2] + G_1 = S + \Delta_S,$$

where $\Delta_S = -A_{21}E_1A_{12} - A_{21}F_1 - F_2 + G_1$. Under the assumption that $\|S^{-1}\Delta_S\| \ll 1$, we then get

$$fl(C_{22}) = fl(fl(S)^{-1}) \approx S^{-1} + \Delta_{S^{-1}},$$

where $\Delta_{S^{-1}} = E_2 - S^{-1}\Delta_S S^{-1}$, and

$$fl(C_{12}) = -((A_{11}^{-1} + E_1)A_{12} + F_1)(S^{-1} + \Delta_{S^{-1}}) + F_3 = C_{12} + \Delta_{C_{12}},$$

where $\Delta_{C_{12}} = (-E_1A_{12} - F_1)S^{-1} - (A_{11}^{-1}A_{12} + E_1A_{12} + F_1)\Delta_{S^{-1}} + F_3$. Dropping second-order terms, we get $\Delta_{C_{12}} \approx A_{11}^{-1}A_{12}\Delta_{S^{-1}} + E_1A_{12}S^{-1} + F_1S^{-1}$. The expression for $fl(C_{21})$ is analogous to that for $fl(C_{12})$. Finally, for C_{11} we get:

$$fl(C_{11}) = (1 - ((C_{12} + \Delta_{C_{12}})A_{21} + F_4))(A_{11}^{-1} + E_1) - F_5 + G_2 = C_{11} + \Delta_{C_{11}}$$

where $\Delta_{C_{11}} = (1 - C_{12}A_{21} - \Delta_{C_{12}}A_{21} - F_4)E_1 - (F_4 + \Delta_{C_{12}}A_{21})A_{11}^{-1} - F_5 + G_2 \approx (1 - C_{12}A_{21})E_1 - (F_4 + \Delta_{C_{12}}A_{21})A_{11}^{-1} - F_5 + G_2$.

Next, we wish to derive upper bounds for the error matrices. To simplify the analysis we only consider one level of recursion, assuming that the lower inverses are computed via LU factorization. This suffices to illustrate the difficulties associated with this algorithm.

Let $\|\cdot\|$ denote an appropriate matrix norm, for example the Frobenius norm or the 2-norm (the exact choice of norm is not important). For matrix addition, we use the error model

$$\|X + Y - fl(X + Y)\| \leq \epsilon \|X + Y\|,$$

while for matrix multiplication, we use the error model

$$\|XY - fl(XY)\| \leq \epsilon \|X\| \|Y\|.$$

The models hold for conventional multiplication with ϵ approximately n times the machine precision (for $n \times n$ matrices), whereas for Strassen's multiplication method ϵ has to be approximately $(n/n_0)^{\log_2 12} n_0^2$ times the machine precision, where n_0 is the size of matrix at which the recursion is terminated [16]. Finally, for matrix inversion we use the following error model, which is certainly reasonable for LU factorization-based methods [15]:

$$\|X^{-1} - fl(X^{-1})\| \leq \epsilon\, \kappa(X)\, \|X^{-1}\|,$$

where $\kappa(A) = \|X\|\,\|X^{-1}\|$ is the condition number of A. We assume that this is a realistic model for ALGORITHM MATINV when not too many levels of recursion are used.

When we apply these relations to the error matrices E_i, F_i and G_i used above, we obtain the following bounds, to first order:

$$\begin{aligned}
\|E_1\| &\leq \epsilon\, \kappa(A_{11})\, \|A_{11}^{-1}\| \\
\|E_2\| &\leq \epsilon\, \kappa(S)\, \|S^{-1}\| \\
\|F_1\| &\leq \epsilon\, \|A_{11}^{-1}\|\, \|A_{12}\| \\
\|F_2\| &\leq \epsilon\, \|A_{21}\|\, \|A_{11}^{-1} A_{12}\| \leq \epsilon\, \|A_{21}\|\, \|A_{11}^{-1}\|\, \|A_{12}\| \\
\|F_3\| &\leq \epsilon\, \|A_{11}^{-1}\, A_{12}\|\, \|S^{-1}\| \leq \epsilon\, \|A_{11}^{-1}\|\, \|A_{12}\|\, \|S^{-1}\| \\
\|F_4\| &\leq \epsilon\, \|C_{12}\|\, \|A_{21}\| \\
\|F_5\| &\leq \epsilon\, \|C_{12}\, A_{21}\|\, \|A_{11}^{-1}\| \leq \epsilon\, \|C_{12}\|\, \|A_{21}\|\, \|A_{11}^{-1}\| \\
\|G_1\| &\leq \epsilon\, \|A_{22}\| + \epsilon\, \|A_{21}\, A_{11}^{-1}\, A_{12}\| \leq \epsilon\, (\, \|A_{22}\| + \|A_{21}\|\, \|A_{11}^{-1}\|\, \|A_{12}\|\,) \\
\|G_2\| &\leq \epsilon\, \|A_{11}^{-1}\| + \epsilon\, \|C_{12}\, A_{21}\, A_{11}^{-1}\| \leq \epsilon\, (\, \|A_{11}^{-1}\| + \|C_{12}\|\, \|A_{21}\|\, \|A_{11}^{-1}\|)
\end{aligned}$$

Inserting these bounds into the expressions for the error matrices Δ_S, $\Delta_{S^{-1}}$, $\Delta_{C_{12}}$ and $\Delta_{C_{11}}$, and neglecting second-order terms, we obtain the following bounds:

$$\|\Delta_S\| \leq \epsilon\, (\, \kappa(A_{11})\, \|A_{21}\|\, \|A_{11}^{-1}\|\, \|A_{12}\| + \|A_{22}\|\,) \tag{3.1}$$

$$\|\Delta_{S^{-1}}\| \leq \epsilon\, \kappa(S)\, \|S^{-1}\| + \|\Delta_S\|\, \|S^{-1}\|^2 \tag{3.2}$$

$$\|\Delta_{C_{12}}\| \leq (\, 2\, \epsilon\, \kappa(A_{11})\, \|S^{-1}\| + \|\Delta_{S^{-1}}\|\,)\, \|A_{11}^{-1}\|\, \|A_{12}\| \tag{3.3}$$

$$\|\Delta_{C_{11}}\| \leq \epsilon\, \kappa(A_{11})\, \|A_{11}^{-1}\|\, (\, 1 + \|C_{12}\|\, \|A_{21}\|\,) + \|\Delta_{C_{12}}\|\, \|A_{21}\|\, \|A_{11}^{-1}\| \tag{3.4}$$

To get a better understanding of these bounds, in particular when ill-conditioning of submatrices is involved, we now make a few simplifying assumptions, namely:

- The matrix A is well conditioned.

- The norms of the submatrices of A satisfy $\|A_{11}\| \approx \|A_{12}\| \approx \|A_{21}\| \approx \|A\|$. (The norm $\|A_{22}\|$ is not important.)
- The norms of the submatrices of $C = A^{-1}$ satisfy $\|C_{22}\| = \|S^{-1}\| \approx \|C_{12}\| \approx \|C\|$.
- The condition number $\kappa(A_{11})$ is much greater than one.

Recall that $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$. Since A_{11} is ill conditioned, its inverse must have large elements, and thus it is likely that the matrix $A_{21}A_{11}^{-1}A_{12}$ completely dominates S (in theory, cancellation could take place and make the elements of $A_{21}A_{11}^{-1}A_{12}$ of the same order as A_{22}, but this is highly unlikely and we ignore this possibility). Hence, we will assume that $\|S\| \approx \|A_{21}\, A_{11}^{-1}A_{21}\| \leq \|A_{12}\|\, \|A_{11}^{-1}\|\, \|A_{12}\|$. Then, since S^{-1} is the (2,2) block of A^{-1},

$$\kappa(S) = \|S\|\, \|S^{-1}\| \lesssim \|A_{21}\|\, \|A_{11}^{-1}\|\, \|A_{12}\|\, \|A^{-1}\| \approx \|A_{11}\|\, \|A_{11}^{-1}\|\, \|A\|\, \|A^{-1}\| = \kappa(A_{11})\, \kappa(A).$$

By means of these and the above bounds, and by neglecting small factors, we then obtain the following approximate bounds for the matrix norms:

$$\|\Delta_S\| \lesssim \epsilon\, \kappa(A_{11})^2\, \|A\| \tag{3.5}$$

$$\|\Delta_{S^{-1}}\| \lesssim \epsilon\, \kappa(A_{11})^2\, \kappa(A)\, \|C\| \tag{3.6}$$

$$\|\Delta_{C_{12}}\| \lesssim \epsilon\, \kappa(A_{11})^3\, \kappa(A)\, \|C\| \tag{3.7}$$

$$\|\Delta_{C_{11}}\| \lesssim \|\Delta_{C_{12}}\|\, \kappa(A_{11}) \lesssim \epsilon\, \kappa(A_{11})^4\, \kappa(A)\, \|C\|. \tag{3.8}$$

Unfortunately, these bounds, which are derived in a quite straightforward manner, are not in accordance with our numerical experiments. Instead, we find experimentally that $\|\Delta_{C_{22}}\|$ and $\|\Delta_{C_{12}}\|$ are approximately proportional to $\kappa(S)$ and that $\|\Delta_{C_{11}}\|$ is approximately proportional to $\kappa(A)^2$. This means that our theoretical bounds are much too pessimistic. In the next subsection, we present our experimental results.

The reason for the lack of success of our simple approach is that we cannot use this approach when cancellation plays an important role in the computations. E.g., even when A_{11} is highly ill-conditioned, $C_{22} = S^{-1}$ cannot have large elements as long as A is well conditioned. Hence, a simple upper bound of the form, say, $\|C_{12}\| \leq \|A_{11}^{-1}\|\, \|A_{12}\|\, \|S^{-1}\| \lesssim \kappa(A_{11})\, \|C\|$ is misleading because we know that $\|C_{12}\| \leq \|C\|$.

3.3 Experimental Error Bounds

We have not been able to derive better theoretical bounds. Instead, we present some numerical results from which we will derive *experimental* bounds for the errors in the computed inverse.

The results are shown in Figure 3.1; see the caption for detailed information. The matrix was generated randomly in such a way that it has an ill-conditioned submatrix A_{11} with a geometric distribution of singular values. The size of A is $n = 64$.

In Figure 3.1, the measured errors ae shown as circles. We see that the errors in the submatrices C_{12} and C_{22} are of the same size and both proportional to $\kappa(S)$ (or $\kappa(S)$), while the errors in the submatrix C_{11} are clearly proportional to $\kappa(S)^2$ (or $\kappa(A_{11})^2$). We found that $\kappa(S)$ was always larger than $\kappa(A_{11})$ and that the following upper bounds—which are shown as crosses in Figure 3.1—are acceptable upper bounds:

$$\|\Delta_{S^{-1}}\| \leq \mathbf{u}\,\kappa(S)\,\|S^{-1}\| \tag{3.9}$$

$$\|\Delta_{C_{12}}\| \leq \mathbf{u}\,\kappa(S)\,\|C_{12}\| \tag{3.10}$$

$$\|\Delta_{C_{11}}\| \leq \mathbf{u}\,\kappa(A_{11})^2\,\|C_{11}\|. \tag{3.11}$$

Here, $\mathbf{u}$ is the machine precision.

Notice that if we insert the first two relations into the theoretical bound for $\|\Delta_{C_{11}}\|$, and if we assume that $\epsilon\,\kappa(A_{11}) < \kappa(A)$, then we obtain $\|\Delta_{C_{11}}\| \lesssim \epsilon\,\kappa(A_{11})\,\kappa(S)\,\|C\|$. However, this is still not in accordance with our observations. Hence, all three experimental bounds above are necessary. We do not know whether these assumptions always hold, or whether worse bounds exist but haven't turned up in our experiments. It may be of interest in the future to apply the direct search methods for matrix computations, described in [17], to this particular problem.

From the above equations we see that the errors in C are dominated by the errors in C_{11}. It is no surprise that $\kappa(A_{11})$ plays a key role here. However, neither the theoretical analysis nor the experimental one makes it clear whether $\kappa(A)$, the condition number of the original matrix, should also appear.

We want to mention here that Strassen's method for computing A^{-1} breaks completely down when the condition number of the submatrix A_{11} exceeds $\mathbf{u}^{-1/2}$.

It is also interesting to consider the residual matrix $R = I - \tilde{A}^{-1}\,A$. Results for the same 10 test matrices as used above are shown in Figure 3.2.

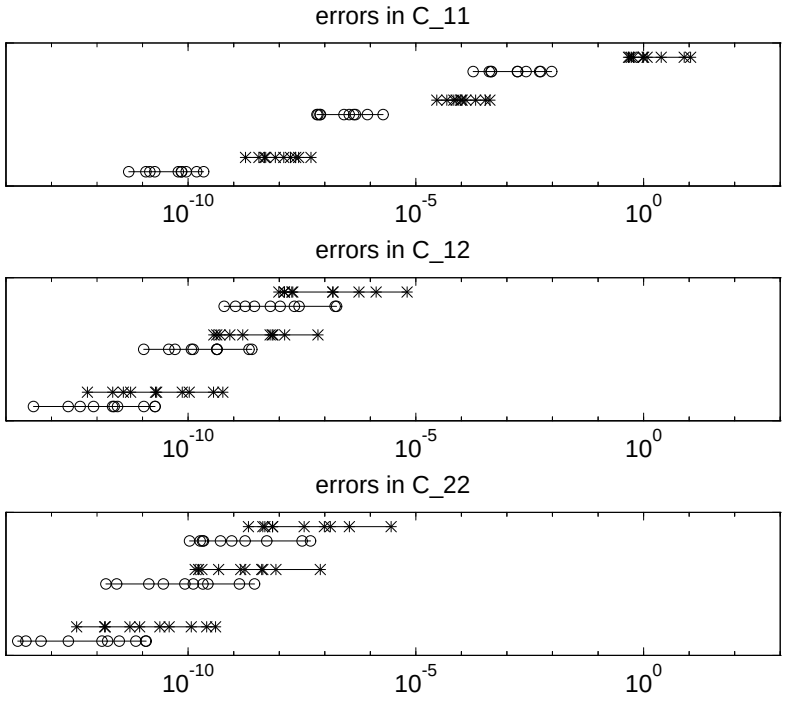

Figure 3.1: Errors in the computed inverse $\tilde{C} = \tilde{A}^{-1}$ as a function of $\kappa(A_{11})$, the condition number of the submatrix A_{11}. The top figure shows results related to the errors in the submatrix C_{11}, while the middle and bottom figures show results related to the errors in C_{12} and C_{22}, respectively. There are ten circles corresponding to 10 experiments. Within each figure, the circles represent measured values of $\|C_{ij} - \tilde{C}_{ij}\|$ for three values of $\kappa(A_{11})$: 10^3 (bottom of figure), 10^5 (middle of figure), and 10^7 (top of figure). Also in each figure, the crosses represent the experimental error bounds mentioned in the text. The condition number of A varies between 10^2 and 10^4, with mean value $1.6 \cdot 10^3$, and with no apparent correlation to $\kappa(A_{11})$.

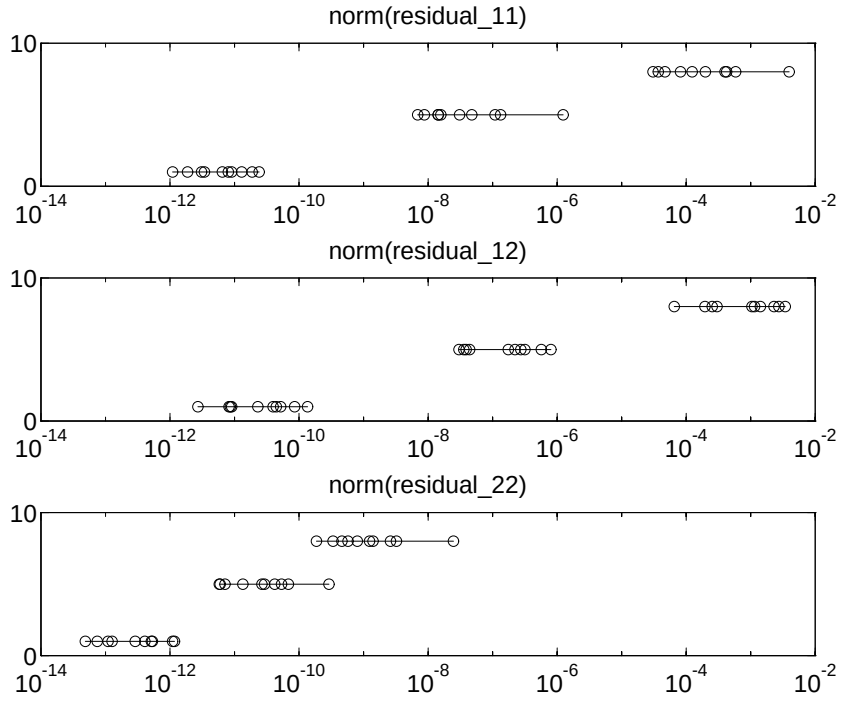

Figure 3.2: The size of the submatrices in the residual matrix $R = I - \tilde{A}^{-1} A$. The top, middle and bottom figures show the norms $\|R_{11}\|$, $\|R_{12}\|$ and $\|R_{22}\|$, respectively, for the same 10 experiments as before. Within each figure, the results are computed for $\kappa(A_{11}) = 10^3$ (bottom), $\kappa(A_{11}) = 10^5$ (middle), and $\kappa(A_{11}) = 10^7$ (top).

We see that $\|R_{22}\|$, the norm of the (2,2) submatrix of R, is of the same order as $\|\Delta_{C_{12}}\|$ and $\|\Delta_{C_{22}}\|$, while both $\|R_{12}\|$ and $\|R_{11}\|$ are of the order $\|\Delta_{C_{11}}\|$. This is no surprise, since we have the relation

$$R = -\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \Delta_{C_{11}} & \Delta_{C_{12}} \\ \Delta_{C_{21}} & \Delta_{C_{22}} \end{pmatrix}$$

from which we conclude that both R_{11} and R_{12} are dominated by $\Delta_{C_{11}}$ while R_{22} must be of the same size as $\Delta_{C_{12}}$ and $\Delta_{C_{22}}$.

4 Stabilization of the Algorithm

In this section we describe our approach to stabilization of the recursive inversion algorithm. As we have seen in the previous section, instability is solely associated with inversion of ill-conditioned submatrices A_{11} and S at any level of the algorithm. In our algorithm, an ill-conditioned submatrix is very easy to detect, since the inverse is explicitly computed, and we can measure the ill-conditioning simply by computing an appropriate estimate of the norm of the inverse matrix (for example, the element of largest absolute value, which can be found in logarithmic time on a parallel computer).

4.1 Diagonal Perturbation

If a submatrix is found to be ill-conditioned, then we add a small perturbation to the submatrix and repeat the inversion process. In the very unlikely event that the perturbed matrix is also ill-conditioned, we make a larger perturbation and try again. We have never encountered this situation in practice.

Perhaps the simplest perturbation that one can use is a diagonal perturbation δI, where δ is a small number. This perturbation has the advantage that it leaves a perturbed Toeplitz matrix in Toeplitz form, which is important in connection with the Bitmead-Anderson algorithm mentioned in the Section 1.

Hence, we are interested in the behavior of perturbations of the form $\tilde{X} = X + \delta I$, where X is a general ill-conditioned matrix. This perturbation merely shifts the eigenvalues of X, but the behavior of the singular values is much more complicated. Stewart [19] has proved that *any* perturbation of a matrix is most likely to *increase* its smallest singular value, and this effect is very pronounced when the matrix has a very small singular, as in our case. Hence, we have good reasons to believe that our heuristic choice of perturbation is indeed a practical one.

Numerical experiments support our choice; we find that the condition of $\tilde{X}$ is always much better than the condition of X. Moreover, when the scalar δ is greater than the smallest singular value of X we always find that the smallest singular value of $\tilde{X}$ is of the order δ.

4.2 The Amount of Stabilization

We can use this information to derive an expression for an adaptive choice of δ. Again, we assume that the submatrix A_{11} is ill-conditioned and that A is well conditioned. We also assume that δ is always chosen to be larger than the smallest singular value of A_{11}, such that $\kappa(\tilde{A}_{11}) \approx \delta^{-1} \|A\|$.

Assume first that we use a very small perturbation. Then rounding errors dominate the process, and we know from the empirical analysis in the previous section that

$$\|C - f\ell(C)\| \approx \|\Delta_{C_{11}}\| \lesssim \epsilon\, \kappa(\tilde{A}_{11})^2\, \|C\| \approx \epsilon\, \delta^{-2}\, \|A\|^2\, \|C\|. \tag{4.1}$$

Next, assume that the perturbation is so large that the perturbation error from adding $\delta\, I$ dominates. Obviously, a perturbation $\delta\, I$ of A_{11} corresponds to a perturbation of A of the form

$$\begin{pmatrix} \delta\, I & 0 \\ 0 & 0 \end{pmatrix}.$$

Then it follows from the perturbation bound

$$\|X^{-1} - (X + \Delta_X)^{-1}\| \lesssim \|\Delta_X\|\, \|X^{-1}\|^2$$

that for large δ we have

$$\|C - f\ell(C)\| \lesssim \delta\, \|C\|^2. \tag{4.2}$$

Both expressions are confirmed by our numerical experiments.

Equating the two upper bounds in (4.1) and (4.2), we then obtain a value of δ for which the two types of perturbations of the computed inverse are balanced:

$$\delta = \sqrt[3]{\epsilon\, \|A\|^2 \,/\, \|C\|^{-1}} = \|A\|\, \sqrt[3]{\delta \,/\, \kappa(A)}. \tag{4.3}$$

Here, we have used that $\kappa(A) = \|A\|\, \|C\|$.

In a practical application, the norm $\|A\|$ (which should really be interpreted as the appropriate submatrix at any given level) is easy to estimate. But estimating the condition number of A is a nontrivial process when no factorization of A is available. The best we can do in connection with this algorithm is probably either to assume some average condition number, say, 1000, or to require the user to provide an rough condition numer estimate (taking the square root means that a very rough guess is enough).

5 Performance Results and Alternative Linear Algebra

In this section we present a few numerical results produced on the 8K Connection Machine CM-200 located at UNI•C. We also make some general comments about linear algebra routines for masively parallel computers.

5.1 Performance on the CM-200

The programs were coded in CM Fortran, Version CMF 1.2, and at the "bottom level" of the recursive algorithm we used the LU factorization algorithm from the CMSSL Library, Version 3.0. The matrix multiplications were carried out using an implementation of Strassen's matrix-matrix multiplication routine, again using a CMSSL routine (for matrix-matrix multiplication) at the bottom level; see [7] for details. No stabilization was added in these timing-tests. Both A and b consist of elements from a rectangular distribution in the interval $[-2, 2]$.

Computation of the approximate inverse is carried out in single precision. The remaining tasks are performed in double precision: $\tilde{x} = \tilde{A}^{-1}b$ is computed, and then $\tilde{x}$ is refined using standard iterative refinement. No extended precision is used in computing the residuals.

Test were carried out with various levels of recursion, and the results are shown in Figure 5.1. Here, Str_x denotes Strassen's methods with x recursion levels, in the sense that Str_1 means that A^{-1} is computed directly from (2.2). The execution times are measured in seconds, and they include the time for a fixed number of 5 steps of iterative improvement. We also show the performance of the CMSSL LU factorization routine, including forward and back substitutions.

We see that the execution time for Strassen's method increases significantly with the number of recursion levels. We believe that the main reason for this is the high number of matrix additions that are required in the recursive matrix-matrix multiplication routine. This is in agreement with the observation that Strassen's matrix-matrix multiplication algorithm seems to perform better on the MasPar MP-1, where multiplication is slower than addition by a factor of three. On the Connection Machines CM-200 and CM-5, however, additions and multiplications are performed at the same speed.

Another important part of the explanation is related to the nearest-neighbor communication. The analysis in [9] shows that a successful imple-

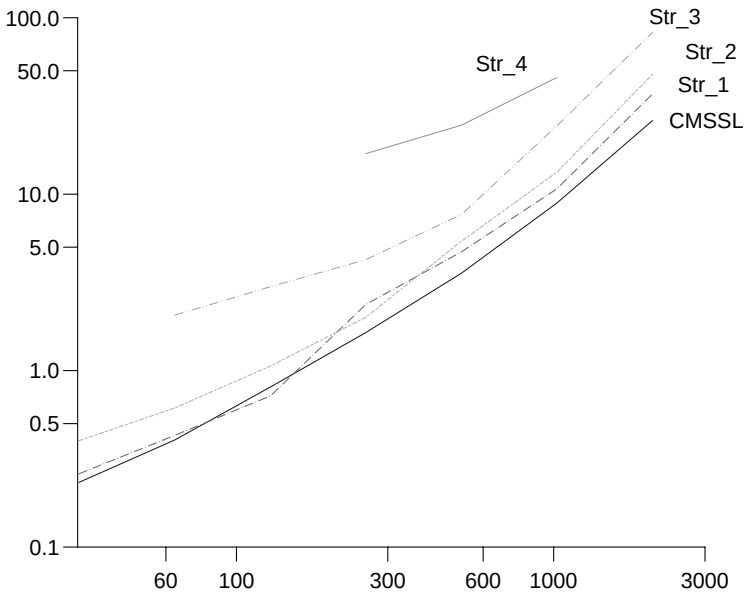

Figure 5.1: Comparison of the performance of Strassen's matrix inversion algorithm (with 1, 2, 3 and 4 recursion levels) with the LU factorization in the CMSSL Library. The figure shows execution times versus matrix size. The times for Strassen's method include 5 steps of iterative refinement, while the times for CMSSL include forward- and back substitution.

mentation of Strassen's algorithm relies on fast nearest-neighbor communication relative to the floating-point operations. If we define γ_X as the time of nearest-neighbor communication divided by the time of an average flop, then $\gamma_{MP-1} = 0.2$ while $\gamma_{CM-200} = 0.4$ and $\gamma_{CM-5} \ll \gamma_{CM-200}$. Again, this shows that Strassen's algorithms may not be useful on the Connection Machines.

It is interesting to notice the following observation that we have made. The amount of mere *communication* can actually be reduced by using the CMSSL matrix-matrix multiplication routine throughout—not just at the bottom level. But then the data must be reorganized before and after each call to this routine, and the performance loss associated with this *reshuffling of data* is greater than the gain in speed from using the high-speed CMSSL multiplication routine.

5.2 Alternative Linear Algebra

The above results are disappointing, but on the other hand they illustrate quite well the difficulties that one faces when developing algorithms and implementing software on today's massively parallel computers.

One of the key issues in connection with today's massively parallel computers is the large amount of time that one must spend in developing efficient implementations on these computers. The development of an efficient LU factorization routine for the CMSSL Library has required several man-years.

In this connection we should also mention that the results presented in [5] (which look more impressive) were based on an older version of the CMSSL Library, where solution of linear equations was performed by a much less "hand-tuned" version of Gauss-Jordan elimination.

This clearly illustrates that "alternative linear algebra routines", such as Strassen's matrix inversion algorithm, may still be important if one is *not* willing—or not able—to put a large effort into hand tuning a certain implementation. ALGORITHM MATINV was implemented with relative ease on the CM-200, and it was able to outperform a standard algorithm because the latter was not carefully hand tuned.

We believe that this is in fact the general picture on massively parallel computers. Here, one seems to be faced with the alternatives:

- either spend a lot of effort on making a very careful implementation, probably based on specialized low-level subroutines,
- or use an alternative algorithm (perhaps based on a modular design),

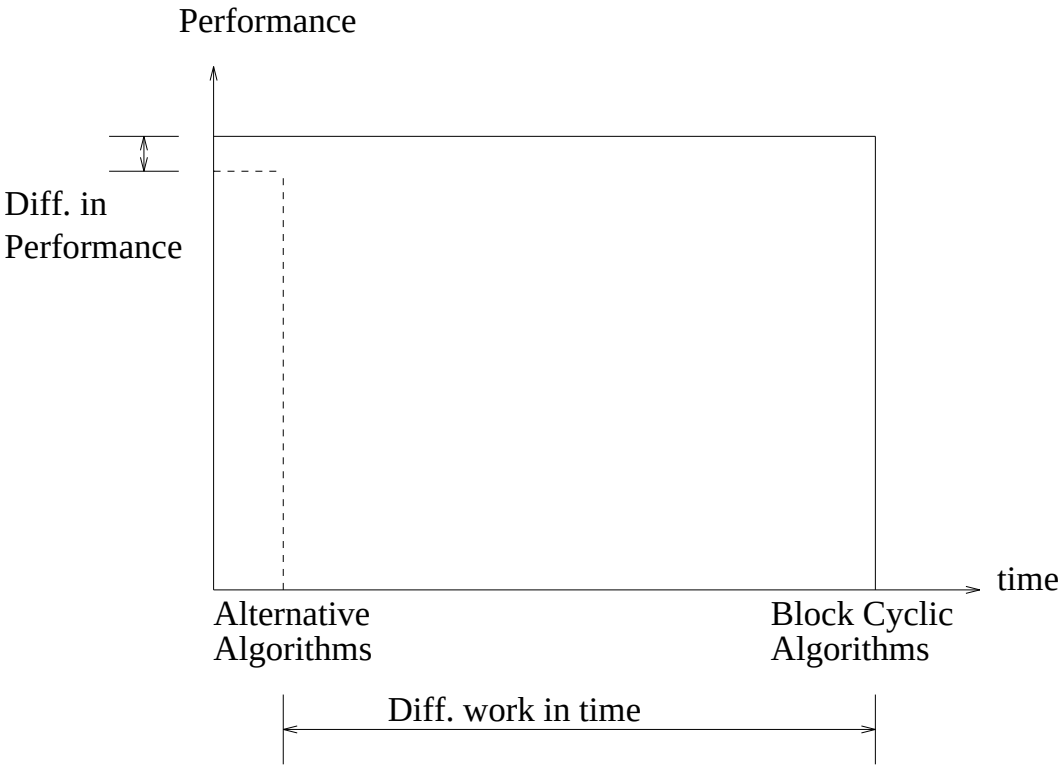

Figure 5.2: Performance versus implementation time for the CM-200.

which can be implemented in much less time, and which performance is acceptable—althoug slower than the "hand-tuned" implementation. It is here that we see on of the applications of alternative linear algebra routines.

Figure 5.2 illustrates this point for the Connection Machines. Here, we know that block cyclic algorithms—when carefully implemented—are probably the algorithms that yield best performance [7]. However, alternative algorithms—such as Strassen's method—give an acceptable performance and can be implemented with much less effort.

6 Final Comments and Future Research

The conclusion of these investigations are quite frustrating.

First, we are not able to perform a satisfactory error analysis of the algorithm. However, the experimental results indicate that the rounding errors are not as harmful as may first be expected, and that with suitable stabilization we are indeed able to produce an approximate inverse which is good enough for iterative refinement or some CG-type method.

Second, the performance of the algorithm on the Connection Machines is disappointing when we compare with the carefully hand-tuned block LU factorization algorithm from the CMSSL Library.

In a continuation of this project (APPARC Deliverable **PaA3**) we seek to address some of the following questions:

1. Consider Strassen's method, as well as block LU factorization, as methods for speeding up a given point algorithm.
2. Look at the Newton-Schulz iteration at the big matrix level
3. Block methods in general on Connection Machines.
4. How fast can LAPACK be made to go by using tuned BLAS or making more extensive changes.

Based upon these investigations, we would like to be able to say more about the trade-off on massively parallel computers between performance and implementation time, as discussed in the previous Section.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchow & D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, 1992.

[2] D. H. Bailey, *Extra high speed matrix multiplication on the Cray-2*, SIAM J. Sci. Stat. Comput. **9** (1988).

[3] D. H. Bailey and H. R. P. Ferguson, *A Strassen–Newton algorithm for high-speed parallelizable matrix inversion.* In *Proceedings of Supercomputing '88*, pages 419–424, New York, 1988. IEEE Computer Society Press.

[4] D. H. Bailey, K. Lee & H. D. Simon, *Using Strassen's algorithm to accelerate the solution of linear systems*, J. Supercomputing **4** (1990).

[5] S. M. Balle & P. C. Hansen, *A Strassen-type matrix inversion algorithm*, in I. Dimov (Ed.), *Workshop on Parallel Algorithms, Sofia, 1992*, World Scientific, to appear.

[6] S. M. Balle & P. C. Hansen, *Block algorithms for the Connection Machines*, Report UNIC-93-08, UNI•C, August 1993.

[7] S. M. Balle, *Solving linear systems on the Connection Machine*, Technical Report in preparation, UNI•C, Denmark, 1993.

[8] R. R. Bitmead & B. D. O. Anderson, *Asymptotically fast solution of Toeplitz and related systems of equation*, Lin. Alg. Appl. **34** (1980), 103–116.

[9] P. Bjørstad, F. Manne, T. Sørevik, & M. Vajteršic, *Efficient matrix multiplication on SIMD computers*, SIAM J. Matrix. Anal. Appl. **13** (1992).

[10] J. W. Demmel,*Trading off parallelism and numerical stability*; en M. S. Moonen, G. H. Golub & B. L. R. De Moor (Eds.), *Linear Algebra for Large Scale and Real-Time Applications*, NATO ASI series, Vol 232, Kluwer, Amsterdam, 1992, 49–68.

[11] J. W. Demmel, M. T. Heath & H. A. Van der Vorst, *Parallel numerical linear algebra*, Acta Numerica (1993), 111–197.

[12] J. W. Demmel and N. J. Higham, *Stability of block algorithms with fast level-3 BLAS*, ACM Trans. Math. Soft., 18(3):274–291, 1992.

[13] J. W. Demmel, N. J. Higham, and R. S. Schreiber. *Block LU factorization*, Numerical Analysis Report No. 207, University of Manchester, England, February 1992; To appear in Journal of Numerical Linear Algebra with Applications.

[14] J. J. Dongarra, I. S. Duff, D. C. Sorensen & H. A. Van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, 1991.

[15] J. J. Du Croz and N. J. Higham, *Stability of methods for matrix inversion*, IMA Journal of Numerical Analysis, 12:1–19, 1992.

[16] N. J. Higham, *Exploiting fast matrix multiplication within the level 3 BLAS*, ACM Trans. Math. Software **16** (1990), 352–368.

[17] N. J. Higham, *Optimization by direct search in matrix computations*, SIAM J. Matrix Anal. Appl., 14(2):317–333, 1993.

[18] W. Lichtenstein & S. L. Johnsson, *Block-cyclic dense linear algebra*, Report, TMC, Boston, (1992); to appear in SIAM J. Sci. Comput.

[19] G. W. Stewart, *A note on the perturbation of singular values*, Lin. Alg. Appl. **28** (1979), 213–216.

[20] V. Strassen, *Gaussian Elimination is not optimal*, Numer.Math. **13**, (1968), 354–356.