A Strassen-Newton Algorithm for High-Speed Parallelizable Matrix Inversion

David H. Bailey
NASA Ames Research Center
Moffett Field, CA 94035

Helaman R. P. Ferguson
Supercomputer Research Center
Lanham, MD 20706

Abstract

This paper describes techniques to compute matrix inverses by means of algorithms that are highly suited to massively parallel computation. In contrast, conventional techniques such as pivoted Gaussian elimination and LU decomposition are efficient only on vector computers or fairly low-level parallel systems.

These techniques are based on an algorithm suggested by Strassen in 1969. Variations of this scheme employ matrix Newton iterations and other methods to improve the numerical stability while at the same time preserving a very high level of parallelism. One-processor Cray-2 implementations of these schemes range from one that is up to 55% faster than a conventional library routine to one that, while slower than a library routine, achieves excellent numerical stability.

The problem of computing the solution to a single set of linear equations is discussed, and it is shown that shown that this problem can also be solved efficiently using these techniques.

Introduction

The problem of efficient machine computation of linear equation solutions and matrix inverses for dense systems was recently considered to be solved. One needed only to apply one of the well-known reduction techniques, such as Gauss-Jordan elimination, LU decomposition, or the like (Press 1986, 20–70). The performances of a wide variety of scientific computers performing such operations are available in listings such as Dongarra's LINPACK results (Dongarra 1987).

However, the recent development of very highly parallel systems, with the prospect of massively parallel systems in the near future, is forcing researchers to take a second look at these techniques. The problem is that traditional schemes do not seem to be conducive to highly parallel computing. For instance, a linear equation solution of an $n \times n$ system using a Gauss-Jordan elimination scheme with partial pivoting requires

1. n searches for the maximum entry in the current column.
2. n interchanges of rows to place the pivot element on the diagonal.
3. n divisions of the row containing the pivot by the pivot element.
4. n reductions of the rows by the pivot row.
5. An unscrambling of the columns of the matrix (if a true inverse is desired).

Of these operations, only item 4 admits any parallelism beyond a single row or column vector (i.e., $O(n)$ processors). Further, the row and column operations are often very short vector operations. Item 1 is singularly ill-suited for highly parallel evaluation. It follows that an implementation of this algorithm on a highly parallel system (i.e., $O(n^2)$ processors) would idle large numbers of processors for a significant fraction of the time.

Adaptations of these traditional algorithms suitable for efficient highly parallel evaluation may eventually be found. However, it is at least as likely that these algorithms are hopelessly unsuited for such systems. In any event, it appears that completely different approaches to this problem need to be considered.

Strassen's Algorithm for Matrix Multiplication

The fact that matrix multiplication can be performed with fewer than $2n^3$ arithmetic operations has been known since 1969, when V. Strassen published an algorithm that asymtotically requires only about $4.7n^{2.807}$ operations (Strassen 1969). Since then other such algorithms have been discovered (Kreczmar 1976, Pan 1980 and 1984). Currently the best known result is due to Coppersmith and Winograd (1987), which reduces the exponent of n to only 2.38. Until recently, however, such advanced algorithms for matrix multiplication were considered to be academic curiosities with no hope of practical application (see for example Press 1986, 74–76). Indeed, it was estimated at the time Strassen's paper was published that the "crossover point" of his algorithm (i.e., the size of a square matrix for which Strassen's algorithm would run faster than the traditional method) was well over 1,000 (Gentleman 1987).

Strassen's algorithm for matrix multiplication is as follows. Let the matrices A, B, and $C = A \times B$ be divided into half-sized blocks:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Then the result may be calculated as follows:

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 &= (A_{21} + A_{22})B_{11} \\ P_3 &= A_{11}(B_{12} - B_{22}) \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})B_{22} \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ C_{11} &= P_1 + P_4 - P_5 + P_7 \\ C_{12} &= P_3 + P_5 \\ C_{21} &= P_2 + P_4 \\ C_{22} &= P_1 + P_3 - P_2 + P_6 \end{aligned}$$

It should be pointed out that the intermediate matrices $P_1, P_2, \cdots, P_7$ may all be computed concurrently. The last four lines may also be computed concurrently, but their cost is generally insignificant compared to the previous seven lines.

The computational savings of employing Strassen's algorithm derives from the fact that only seven half-sized matrix multiplications need to be performed, whereas eight are required with the standard algorithm. Thus for fairly large matrices, where the cost of performing this algorithm is dominated by the cost of multiplying the half-sized blocks, a savings of approximately 14% can be realized (in theory) over the traditional method.

Strassen's method can of course be recursively employed to multiply the half-sized blocks, and so on down to individual matrix elements if desired. For every level of recursion, an additional 14% savings can be realized. However, in practice this recursion is only performed down to the level at which the losses due to bookkeeping costs and short vector lengths overwhelm any savings due to fewer floating-point operations being performed. On the Cray-2, for example, this crossover point was found to be for matrices of size 128 × 128.

Each level of recursion also multiplies the number of concurrently executable tasks by seven. Thus when even as few as four levels of recursion are employed, thousands of independent tasks are generated. Once tasks are smaller than the crossover point, each individual task still has substantial amounts of parallelism. For example, using the traditional scheme for matrix multiplication, each of the 16,384 result elements of a 128 × 128 operation can be computed concurrently. Therefore this method has almost unlimited parallelism; it is apparently limited only by the capacity of the particular interconnection network to support the necessary data movement (Gentleman 1978).

Other details of practical implementations of Strassen's matrix multiplication algorithm may be found in Bailey (1987).

Strassen's Algorithm for Matrix Inversion

Not nearly as well known as Strassen's algorithm for matrix multiplication is a related algorithm for matrix inversion, which appeared in the same paper. This algorithm can be described as follows. Let the matrices A and $C = A^{-1}$ be divided into half-sized subblocks:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Then the result C may be calculated as follows:

$$\begin{aligned} P_1 &= A_{11}^{-1} \\ P_2 &= A_{21} \times P_1 \\ P_3 &= P_1 \times A_{12} \\ P_4 &= A_{21} \times P_3 \\ P_5 &= P_4 - A_{22} \\ P_6 &= P_5^{-1} \\ C_{12} &= P_3 \times P_6 \\ C_{21} &= P_6 \times P_2 \\ C_{11} &= P_1 - P_3 \times C_{21} \\ C_{22} &= -P_6 \end{aligned}$$

These steps, for the most part, cannot be performed concurrently. However, for a large matrix there is enough work within each step to admit massively parallel computation. In particular, the computation required to perform this algorithm at a certain level is dominated by the cost of the six matrix multiplications, which can of course be performed by Strassen's matrix multiplication algorithm. The two matrix inversion operations in the above can be performed by recursive application of this same method. It can easily be seen that with just a few levels of recursion, almost all real work is square matrix multiplication.

That this technique is faster than conventional techniques even in a one-processor implementation can be seen in Table 1, which contains some Cray-2 performance figures for matrices containing Gaussian pseudorandom data. In this table, the columns headed "Cray Library" contain figures for Cray's MINV routine. MINV is coded in assembly language and is very highly optimized for the Cray-2 — it runs at 300 MFLOPS or more on a single processor. The columns headed "Strassen" contain figures for a recursive implementation of Strassen's algorithm as above, with matrices of sizes 128 × 128 or smaller performed with MINV. The timings shown are the means of ten runs on a single processor, with a normal background of other jobs on the three remaining processors. The error statistics are the geometric means of the root-mean-square (RMS) errors for the ten runs. The RMS error E for a single trial is computed as

$$E = \frac{1}{n}\sqrt{\sum_{i,j}(B_{ij} - I_{ij})^2}$$

where B is the product of the computed inverse with the original matrix and I is an $n \times n$ identity matrix.

Actually, the results in the Cray Library columns were obtained by using a special version of MINV that does not perform determinant calculations. For these large matrices, even the extremely large dynamic range of the Cray-2 ($10^{\pm 2,465}$) is insufficient, and floating-point overflows result when determinants are calculated. In addition to this modification, a matrix to be inverted by MINV is first copied into an array with an odd first dimension. This avoids the substantial performance loss that otherwise results from bank conflicts when a matrix whose size is divisible by a power of two is accessed by rows.

While the timings listed in Table 1 are very encouraging, the error statistics indicate that the Strassen routine is not as numerically stable as MINV, which employs Gauss-Jordan elimination with partial pivoting. Indeed, for large matrices the Strassen inverse scheme is only accurate to about seven digits (on the average), even when full 14-digit arithmetic is employed. This may be sufficient for some applications known to be highly stable, but for other applications it is unsatisfactory.

A Pivoted Version of the Strassen Matrix Inverse Algorithm

The principal reason for the mediocre accuracy statistics of the Strassen inverse scheme is that the algorithm as described above first attempts to invert the upper left $n/2 \times n/2$ matrix. If this block is well-conditioned, then a fairly accurate full-sized inverse will be produced. If this block is near-singular, then the accuracy of the full-sized inverse will be poor. If this block is singular, the method will fail.

This analysis suggests that a pivoted version of Strassen's scheme would exhibit greater stability. Indeed, if the full-sized matrix is well-conditioned, then either the upper left corner block or the lower left corner block must be well-conditioned. If the lower left corner block is well-conditioned, then the inverse of the full-sized matrix may be computed by a variant of the Strassen matrix inverse algorithm, as follows:

$$\begin{aligned}
P_1 &= A_{21}^{-1} \\
P_2 &= A_{11} \times P_1 \\
P_3 &= P_1 \times A_{22} \\
P_4 &= A_{11} \times P_3 \\
P_5 &= P_4 - A_{12} \\
P_6 &= P_5^{-1} \\
C_{12} &= P_3 \times P_6 \\
C_{21} &= P_6 \times P_2 \\
C_{11} &= P_1 - P_3 \times C_{22} \\
C_{22} &= -P_6
\end{aligned}$$

This suggests the following alternate scheme for matrix inversion: compute the inverse and determinant of both the upper left and lower left blocks, and proceed with the variant of the Strassen algorithm corresponding to the block with the largest determinant. The determinant of a block may be easily computed along with the inverse as a part of either variant of Strassen's inverse algorithm, as follows. First note that

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

As a result, one can calculate

$$\det(A_{11}^{-1}) \det \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \det(A_{22} - A_{21}A_{11}^{-1}A_{12})$$

Thus it follows that

$$\det \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \det(A_{22} - A_{21}A_{11}^{-1}A_{12}) \det(A_{11})$$

By a happy coincidence, the two matrices indicated in the last line are exactly equal (except for sign) to the two matrices whose inverses are computed in the first variant of the Strassen inverse algorithm (see the formulas for P_1 and P_6). A similar result holds for the second variant. The determinant of the $n/2 \times n/2$ block can be calculated by recursive application of this same procedure.

At the bottom level of recursion, Cray's MINV can be used to obtain both the inverse and determinant, since calculation of the determinant of a 128 × 128 or smaller block does not overwhelm the machine precision. One way to avoid floating-point overflow in the recursive calculation of larger determinants is to take the square root of the product of the determinants before passing the result to a higher level.

While the determinant is for most problems a satisfactory measure of condition, other easily computed statistics can also be used. For example, a simple enhancement of the determinant calculation is to normalize it by the maximum entry of the block, or by the sum of the absolute values of the entires. This prevents such pathologies as a block consisting of a small constant times the identity being rejected because its determinant is small.

Employing Newton Iterations to Enhance Precision

The precision of the results may be further enhanced by employing matrix Newton iterations. A quadratic Newton iteration is defined as

$$X_{k+1} \quad = \quad (2I_n - X_k A) X_k$$

where X_k is an approximation to the inverse of A. It can be easily shown that for suitable choices of a starting matrix X_0, the iterates X_k converge quadratically to A (Pan and Reif 1985). This means that each iteration approximately doubles the number of correct digits in the entries of X_k. Higher order Newton iterations can also be defined that exhibit m-th order convergence (Ferguson 1987). In the following, however, quadratic Newton iterations will be implied.

One very attractive feature of matrix Newton iterations is that they are massively parallelizable, since they involve only matrix multiplications and matrix subtractions. The matrix multiplications can of course be efficiently performed using Strassen's algorithm. However, the total cost of computation is quite high. One Newton iteration applied to a full-sized matrix alone requires twice the number of floating-point operations as is normally required to compute the inverse using a classical technique such as Gauss-Jordan elimination. Nonetheless, its suitability for parallel processing may well overcome this disadvantage on a massively parallel system.

There are several ways that Newton iterations can be employed in conjunction with Strassen's inversion algorithm. One is to simply apply one Newton iteration to the final result of a pivoted Strassen scheme. Another is to apply a Newton iteration to the result at each level of recursion except the highest level. As it turns out, this increases the total computational requirement only fractionally, yet it significantly increases the accuracy of the result. Finally, one can apply a Newton iteration to each level of recursion including the highest level.

Tables 2 and 3 contain detailed results of Cray-2 runs employing the above schemes. The column headed "MINC" contains results for Cray's MINV, with the modifications described above (i.e. no determinant calculation, and the input matrix is copied to an array with an odd first dimension before referencing MINV). MINS is a "no-frills" implementation of Strassen's basic matrix inverse algorithm. MINX is the pivoted version, employing both variants of the Strassen algorithm and determinant calculations. MINY in addition employs one Newton iteration at each level of recursion (except the highest level) to refine the accuracy of the intermediate results. MINZ also employs a Newton iteration at the highest level. As before, the CPU times are the means of the run times (in seconds) of ten trials, and the error statistics are the geometric means of the RMS errors of the ten trials.

It can be seen from the table that both MINS and MINX have faster run times than MINC. Even MINY, which employs Newton iterations at all levels except the highest, still is only slightly slower than MINC. MINZ requires more than twice as much CPU time (on a single processor) as MINC. The accuracy statistics, however, indicate a steady progression of improvement among the four new schemes. MINY is almost as good as MINC, and MINZ appears to be on a par with or slightly better than MINC.

Solution of a Single Set of Linear Equations

One objection often raised in the discussion of matrix inversion is that most applications of matrix techniques only require the solution of a single set of linear equations. For this case standard reduction methods can produce the solution in about one third the time required for a full matrix inversion followed by matrix-vector multiplication.

However, even this problem can be efficiently solved by the application of the above techniques. Let $Ax = b$ denote a set of linear equations. Let $A_{11}, A_{12}, A_{21}, A_{22}$ be the four half-sized sections of A, and let x_1, x_2, b_1, b_2 be corresponding halves of x and b. Then the system

Matrix	Cray Library		Strassen		Time	Error
Size	CPU Time	Error	CPU Time	Error	Ratio	Ratio
128	0.0214	1.50E-13	0.0231	1.50E-13	0.926	1.00E 0
200	0.0796	2.87E-13	0.0652	4.58E-11	1.221	6.26E-3
256	0.1373	1.69E-13	0.1208	6.28E-12	1.137	2.69E-2
400	0.5394	3.37E-13	0.4372	3.95E-10	1.234	8.53E-4
512	1.0498	3.91E-13	0.7866	3.68E-09	1.335	1.06E-4
800	3.8273	9.46E-13	2.9879	6.87E-09	1.281	1.38E-4
1024	8.1224	8.55E-13	5.5440	1.40E-07	1.465	6.10E-6
1600	30.0079	1.67E-12	22.0779	1.53E-07	1.359	1.10E-5
2048	62.0103	1.52E-12	39.9997	3.59E-07	1.550	4.24E-6

Table 1: Cray-2 Strassen Algorithm Performance Results

Size	MINC	MINS	MINX	MINY	MINZ
128	0.0214	0.0231	0.0205	0.0216	0.0223
200	0.0796	0.0652	0.0748	0.0739	0.1709
256	0.1373	0.1208	0.1315	0.1377	0.3159
400	0.5394	0.4372	0.5060	0.6969	1.4261
512	1.0498	0.7866	0.9227	1.3203	2.5881
800	3.8273	2.9879	3.6127	5.7284	10.7166
1024	8.1224	5.5440	6.6607	10.5752	19.6775
1600	30.0079	22.0779	25.9482	43.1163	78.9089
2048	62.0103	39.9997	47.8864	79.4194	146.5576

Table 2: Cray-2 CPU Times for the Five Algorithms

Size	MINC	MINS	MINX	MINY	MINZ
128	1.497E-13	1.497E-13	1.497E-13	1.497E-13	1.497E-13
200	2.869E-13	4.584E-11	8.341E-12	8.341E-12	1.192E-13
256	1.687E-13	6.277E-12	6.503E-12	6.503E-12	6.253E-14
400	3.371E-13	3.952E-10	1.085E-10	1.769E-11	1.944E-13
512	3.914E-13	3.679E-09	1.082E-10	1.217E-11	1.964E-13
800	9.462E-13	6.870E-09	4.315E-09	2.757E-10	7.626E-13
1024	8.551E-13	1.402E-07	4.677E-09	9.911E-11	6.030E-13
1600	1.674E-12	1.529E-07	7.343E-08	8.913E-10	1.937E-12
2048	1.522E-12	3.587E-07	5.280E-08	1.778E-09	1.535E-12

Table 3: Cray-2 Error Statistics for the Five Algorithms

may be solved as follows:

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 &= b_1 \\ A_{21}x_1 + A_{22}x_2 &= b_2 \\ x_1 &= A_{11}^{-1}(b_1 - A_{12}x_2) \\ (A_{22} - A_{21}A_{11}^{-1}A_{12})x_2 &= b_2 - A_{21}A_{11}^{-1}b_1 \end{aligned}$$

and this system may be solved by recursively applying the same algorithm. Once x_2 is obtained, x_1 can be computed from the third line. Note that at a given level of recursion, the only items that have significant cost are the two matrix multiplications and one matrix inversion in the left-hand side of the last line. These operations may be efficiently performed by applying Strassen's matrix multiplication algorithm and one of the matrix inversion algorithms discussed above. The expression on the right hand side of the last line has virtually no cost for large n, since its only operations are one matrix subtraction and one matrix-vector multiplication. Similarly, the cost of computing x_1 once x_2 is obtained is insignificant when n is large. The overall cost of solving a linear system using this method is only slightly more than one third of the cost of inverting the full-sized matrix.

Conclusion

Four variations of Strassen's matrix inversion have been described, each of which is highly suited for massively parallel computation. Each of the four has different CPU time/accuracy characteristics. They range from a routine that is 55% faster than traditional methods, even on a one-processor system, to a routine that achieves excellent numerical stability in addition to retaining its suitability for massively parallel computation. These methods can also be applied to obtain an efficient solution of a single linear system.

There are obviously many more combinations of these techniques, some of which have been suggested in this paper. Higher order Newton iterations could also be explored, although it did not appear that the higher order methods would be cost-effective for the variations tried in this study.

Due to the very high level of parallelism inherent in these algorithms, it is believed that they will be particularly effective on highly parallel systems. Clearly the only way to decide such issues is to try such techniques on highly parallel systems and compare them with conventional matrix techniques. It is hoped that some parallel experiments of this sort can be performed during the next year.

REFERENCES

1. Bailey, David H., "Extra-High Speed Matrix Multiplication on the Cray-2", *SIAM Journal on Scientific and Statistical Computing*, to appear, manuscript dated September 1987.
2. Coppersmith, D., and Winograd, S., "Matrix Multiplication via Arithmetic Theory", *19th ACM Symposium on Theory of Computing*, 1987, p. 1 - 6.
3. Dongarra, Jack J., "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment", Argonne Technical Memorandum No. 23, October 27, 1987.
4. Ferguson, H.R.P., Bailey, D.H., and Galway, W.F., "A Numerically Stable, High-Precision, Parallelizable Matrix Inversion Algorithm", manuscript dated January 1987.
5. Gentleman, W. Morven, "Some Complexity Results for Matrix Computations on Parallel Processors", *Journal of the ACM*, Vol. 25 (1978), p. 112-115.
6. Gentleman, W. Morven, private communication, March 1987.
7. Kreczmar, A., "On Memory Requirements of Strassen Algorithms", in *Algorithms and Complexity: New Directions and Recent Results*, J. F. Traub, ed., Academic Press, New York, 1976.
8. Kronsjo, Lydia, *Computational Complexity of Sequential and Parallel Algorithms*, John Wiley, 1985.
9. Pan, V., "New Fast Algorithms for Matrix Operations", *SIAM Journal on Computing*, Vol. 9 (1980), p. 321-342.
10. Pan, V., "How Can We Speed Up Matrix Multiplication?", *SIAM Review*, 26 (1984), p. 393-415.
11. Pan, V., and Reif, J., "Efficient Parallel Solution of Linear Systems", *ACM Symposium on Theory of Computing*, May 1985.
12. Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recipes*, Cambridge University Press, New York, 1986.
13. Strassen, V., "Gaussian Elimination Is Not Optimal", *Numerical Mathematics*, Vol. 13 (1969), p. 354-356.