Extra-High Speed Matrix Multiplication on the Cray-2

David H. Bailey

September 2, 1987

Ref: *SIAM J. on Scientific and Statistical Computing*, vol. 9, no. 3, (May 1988), pg. 603–607

Abstract

The Cray-2 is capable of performing matrix multiplication at very high rates. Using library routines provided by Cray Research, Inc., performance rates of 300 to 425 MFLOPS can be obtained on a single processor, depending on system load. Considerably higher rates can be achieved with all four processors running simultaneously.

This article describes how matrix multiplication can be performed even faster, up to twice the above rates. This can be achieved by (1) employing Strassen's matrix multiplication algorithm to reduce the number of floating-point operations performed and (2) utilizing local memory on the Cray-2 to avoid performance losses due to memory bank contention. The numerical stability and potential for parallel application of this procedure are also discussed.

The author is with the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center, Moffett Field, CA 94035.

Introduction

Recently a number of high-performance multi-processor vector computers have become available for scientific computation. One of these is the Cray-2, manufactured by Cray Research, Inc. It features over 268 million 64-bit words of main memory, which is between one and two orders of magnitude more than that provided in previous generations of supercomputers. In addition to this large main memory, the Cray-2 features four central processing units (CPUs), each with a hardware clock period of only 4.1 nanoseconds, as compared to 9.5 nanoseconds on the Cray X-MP line. Thus each Cray-2 CPU is potentially about twice as fast as a Cray X-MP CPU. However, performance rates on typical vectorized FORTRAN programs are only about on a par with the Cray X-MP, largely due to the fact that chips commensurate in speed with the fast Cray-2 CPUs were not available when the first few Cray-2 systems were manufactured.

For applications that can effectively utilize certain assembly-coded library routines, however, the higher power of the Cray-2 can be harnessed. In particular, Cray's library routine for matrix multiplication (MXM) is capable of speeds that are close to the peak hardware speeds. For one processor running without memory interference from other processors, a performance rate of 425 million floating-point operations per second (MFLOPS) has been achieved on a matrix multiplication in a stand-alone environment. Even with the other processors busy as in a normal operating environment, rates near 300 MFLOPS can easily be achieved. Using a four processor, multi-tasked version of this routine, over 1700 MFLOPS has been achieved.

Can matrices be multiplied on the Cray-2 significantly faster than this? Yes. In fact, large matrices can be multiplied with more than twice the speed of the MXM routine. Clearly such speedups cannot be obtained solely by more efficient implementation of the usual matrix multiplication scheme, although some improvement can be made in this area. The key to such speedups is to employ an advanced algorithm that produces the matrix product using fewer floating-point operations, while still maintaining a high level of vectorization and functional unit concurrency.

Strassen's Matrix Multiplication Algorithm

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 [1]. Since then other such algorithms have been discovered [2], and currently the best known result is due to Coppersmith and Winograd [3], which reduces the exponent of n to only 2.496.

These more recently discovered algorithms are considerably more complicated than Strassen's algorithm and do not significantly improve upon Strassen's algorithm unless the matrices are quite large (i.e., $1,000 \times 1,000$ or so). Thus this article will focus on the implementation of Strassen's algorithm, which is as follows:

Let the matrices A, B, and C 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. In any event Strassen's algorithm appears to be fairly well suited for multi-processor computation.

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 this crossover point was found to be for matrices of size 128×128.

Reducing Memory Bank Contention

The Cray-2 has 128 independent, interleaved main memory banks. Thus a vector fetch of 64 contiguous data words obtains each word from a separate bank, provided each of the requested banks is not busy. However, with four CPUs actively accessing main memory, the probability that a requested bank will be busy is fairly high. As a result, the overall performance of the Cray-2 on memory intensive programs such as matrix multiplication is

reduced as much as 35% below what it would be in the absence of contention. This loss will be reduced in the future as faster memory chips become available.

One way to avoid this performance loss is to employ the 16,384 word local memory in each CPU to perform block matrix multiplications. In this way main memory traffic is sharply reduced, and the CPU computational units can perform nearly at peak speed, even when the system is busy with jobs on the other CPUs. Don Calahan of the University of Michigan has prepared an assembly-coded matrix multiplication routine based on this principle, and this routine was employed in these calculations.

Performance Results

For testing purposes, Strassen's algorithm was coded in FORTRAN for square matrices, although it should not be difficult to generalize the implementation for non-square matrices since the Strassen formulas still hold for this case. Calahan's local memory routine was referenced for matrices smaller than 128×128. Recursion was obtained merely by replicating the FORTRAN subroutine and changing the name at each level of recursion. Although little real work was performed in the FORTRAN subroutines, these operations were vectorized by the FORTRAN compiler. No attempt was made to utilize more than one of the four Cray-2 CPUs. However, if this were done, speedups of nearly four times should be possible because inter-processor memory bank contention is minimized by the use of Calahan's routine.

The usual implementation of Strassen's routine requires additional memory. In this program, a scratch array of size $3n^2$ was required in addition to the input and result arrays. On a large memory computer such as the Cray-2, memory space is not a serious issue, and in fact trading memory space for increased performance is an effective use of the Cray-2. On systems where memory is more dear, a version proposed by Kreczmar [4] may be preferable. That version reduces the extra storage requirements to $2n^2/3$.

Table 1 compares the performance of this implementation with that of the Cray library matrix multiplication routine. Columns headed "Strassen Routine" contain data for the technique described above, while "Cray Library MXM" contain data for the Cray library matrix multiplication routine. These results are based on ten trials with pseudorandom normally distributed data. The error statistic reported is the relative root-mean-square error. To be precise, the error statistic E is given by

$$E = n^{-3/2} \sum_{i,j} (A_{ij} - S_{ij})^2$$

where A is the computed matrix product and S is exact matrix product. The exact matrix S was computed using double-precision calculations, which are feasible only up to $1,024 \times 1,024$. Cray computers do not have hardware for performing arithmetic on double precision (128 bit) data, and as a result double precision operations are two orders of magnitude slower than single precision arithmetic.

It can be seen from the results in the table that the new routine is approximately 35% faster than MXM for small matrices. This speedup is entirely due to Calahan's local

Matrix	Strassen Routine		Cray Library MXM		Time
Size	CPU Time	Error	CPU Time	Error	Ratio
64	0.0014	9.686×10^{-15}	0.0019	9.686×10^{-15}	1.35
100	0.0057	9.187×10^{-15}	0.0078	1.214×10^{-14}	1.35
128	0.0112	1.003×10^{-14}	0.0162	1.355×10^{-14}	1.45
200	0.0474	1.891×10^{-14}	0.0636	1.704×10^{-14}	1.34
256	0.0881	1.999×10^{-14}	0.1401	1.915×10^{-14}	1.59
400	0.3548	3.745×10^{-14}	0.4844	2.420×10^{-14}	1.37
512	0.6452	3.997×10^{-14}	1.0473	2.730×10^{-14}	1.62
800	2.5878	7.484×10^{-14}	3.7262	3.424×10^{-14}	1.44
1024	4.7107	7.993×10^{-14}	8.7800	3.882×10^{-14}	1.86
1600	18.3251		28.3005		1.54
2048	33.1119		66.6682		2.01

Table 1: Comparative Matrix Multiplication Performance

memory matrix multiplication routine. Beginning at size 128×128, a larger speedup is obtained, indicating the positive effect of the Strassen algorithm. Some irregularity can be seen in the speedup figures from entry to entry, but these figures are monotonic if restricted to powers of two or to non-powers of two. For a $2,048 \times 2,048$ matrix multiplication, which was the largest case studied, a speedup factor of 2.01 was obtained. Again, all but 35% of this speedup is due to usage of the Strassen algorithm.

Numerical Stability of Strassen's Algorithm

It can be seen from the figures in the table that the numerical errors in Strassen's algorithm, although slightly larger than the ordinary inner product method, appear to be well under control in the cases studied. In general Strassen's algorithm is known to be not as numerically stable as the inner product calculation, although it still satisfies an acceptable stability condition.

In 1975 Webb Miller [5] showed that any scheme satisfying a sufficiently strong notion of stability would have to perform at least n^3 multiplications to evaluate a $n \times n$ matrix product. Thus the ordinary inner product method is optimal for this stability condition. However, Strassen's algorithm is known [5] to satisfy a condition known as simultaneous Brent stability. For our purposes these two stability conditions can be defined as follows. Let $c_{ik} = \sum_j a_{ij} b_{jk}$ denote the usual inner product of two matrices A and B. Let Δc_{ik} denote the numerical error in calculating c_{ik} by some particular procedure (this term is more precisely defined in Miller's paper). Then simultaneous Brent stability and simultaneous strong stability are defined as, respectively,

$$\Delta c_{ik} \leq C(\max_j |a_{ij}|)(\max_j |b_{jk}|) \quad \text{for every } i, k$$

$$\Delta c_{ik} \leq C \sum_j |a_{ij} b_{jk}| \quad \text{for every } i, k$$

An example where Strassen's method is not strongly stable is as follows. Let ϵ denote a value on the order of the machine "epsilon" (i.e., 2^{-b}, where b is the number of mantissa bits in the representation of floating-point numbers). Consider the 2×2 matrix product

$$\begin{bmatrix} \epsilon^2 & 1 \\ 1 & \epsilon \end{bmatrix} \begin{bmatrix} 1 & \epsilon \\ \epsilon^2 & 1 \end{bmatrix} = \begin{bmatrix} 2\epsilon^2 & 1+\epsilon^2 \\ 1+\epsilon^3 & 2\epsilon \end{bmatrix}$$

Note that c_{11} is of order ϵ^2. In performing Strassen's algorithm to evaluate this product, c_{11} is computed as

$$c_{11} = 2\epsilon(1+\epsilon) - \epsilon(1-\epsilon^2) - (1+\epsilon^2) + (1-\epsilon)(1+\epsilon^2)$$

Because this calculation adds and subtracts numbers of order unity, the numerical error in calculating c_{11} is potentially of order ϵ. However, both terms of the sum in the strong stability condition for c_{11} above are of order ϵ^2. Thus the strong stability condition can fail by an unbounded factor — the greater the level of machine precision, the greater the potential error factor.

In other words, these results imply that matrix products computed using the Strassen algorithm can only be relied on to a level of accuracy that is on the order of the machine epsilon times the largest value of the matrix. For most applications, this degree of accuracy is completely acceptable — nothing more is required of a linear equation solution, for example. Certainly away from a set of matrices of small measure there is no problem whatsoever, other than a somewhat faster accumulation of error due to the increased number of additions and subtractions that are part of Strassen's algorithm.

Conclusion

Strassen's algorithm appears to be a practical means of accelerating matrix multiplication. It produces a significant speedup of this operation whenever the matrices are sufficiently large to overcome the effects of bookkeeping costs and the shorter vector length of block matrix operations at the base level. These requirements should be met for matrices of reasonable size on a variety of currently available scientific computer systems. Thus this algorithm should be considered by anyone wishing to implement a high-performance matrix multiply routine for scientific computation.

REFERENCES

1. Strassen, V., "Gaussian Elimination Is Not Optimal", *Numerical Mathematics*, Vol. 13 (1969), p. 354-356.

2. Pan, V., "New Fast Algorithms for Matrix Operations", *SIAM Journal on Computing*, Vol. 9 (1980), p. 321-342.

3. Coppersmith, D., and Winograd, S., "On the Asymptotic Complexity of Matrix Multiplication", *SIAM Journal on Computing*, Vol. 11 (1982), p. 472-492.

4. 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.

5. Miller, Webb, "Computational Complexity and Numerical Stability", *SIAM Journal on Computing*, Vol. 4 (1975), p. 97-107.

6. Kronsjo, Lydia, *Computational Complexity of Sequential and Parallel Algorithms*, John Wiley, 1985.