Variants of matrix–matrix multiplication for Fortran–90

Craig C. Douglas[1] and **Gordon Slishman**[1]

Abstract

The Fortran–90 standard requires an intrinsic function *matmul* which multiplies two matrices together to produce a third as the result. However, the standard does not specify which algorithm to use. We consider an extension to the *matmul* syntax which allows a Winograd variant of Strassen's algorithm to be added. We discuss an implementation that is in a commercial Fortran–90 offering.

Key words. BLAS, matrix multiplication, Winograd's variant of Strassen's algorithm, multilevel algorithms

AMS(MOS) subject classification. Numerical Analysis: Numerical Linear Algebra

1 Introduction

Fortran–90 [1] has a rich set of intrinsic functions built into it that operate on large objects such as vectors and matrices. We would like to draw attention to the fact that the algorithms that implement these functions are not usually specified in the standard. As a result of this, IBM™ recently added a Winograd variant of Strassen's algorithm in its Fortran–90 compiler, XLF 3.1.1 (the April, 1994 update [5]). This compiler is currently available on the Internet by anonymous ftp from *software.watson.ibm.com* in the directory *pub/aix3/xlf.* While the default is to use the classical algorithm, a simple change to a program results in much higher performance, though with a risk.

Suppose we want to multiply two matrices

$$A : M \times K \quad \textit{and} \quad B : K \times N,$$

where M, K, and N are natural numbers.

Strassen's method recursively works with sets of 2×2 submatrices to form the product using 7 matrix multiplications instead of the obvious 8. This is not very different from standard multilevel methods [2] used routinely to solve partial differential equations. Strictly speaking, we compute

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

using the following algorithm (*Strassen–Winograd*):

$$\left\{ \begin{array}{lcl} S_7 & = & B_{22} - B_{12} \\ S_3 & = & A_{11} - A_{21} \\ M_4 & = & S_3 S_7 \\ S_1 & = & A_{21} + A_{22} \\ S_5 & = & B_{12} - B_{11} \\ M_5 & = & S_1 S_5 \\ S_6 & = & B_{22} - S_5 \\ S_2 & = & S_1 - A_{11} \\ M_1 & = & S_2 S_6 \\ S_4 & = & A_{12} - S_2 \\ M_6 & = & S_4 B_{22} \\ T_3 & = & M_5 + M_6 \\ M_2 & = & A_{11} B_{11} \\ T_1 & = & M_1 + M_2 \\ C_{12} & = & T_1 + T_3 \\ T_2 & = & T_1 + M_4 \\ S_8 & = & S_6 - B_{21} \\ M_7 & = & A_{22} S_8 \\ C_{21} & = & T_2 - M_7 \\ C_{22} & = & T_2 + M_5 \\ M_3 & = & A_{12} B_{21} \\ C_{11} & = & M_2 + M_3 \end{array} \right. \qquad (1)$$

(see [3], [6], and [7]).

The principal advantage of (1) is that it is an $O(n^{2.81})$ algorithm (for square matrices of order n),

[1] Mathematical Sciences Department, IBM Research Division, Thomas J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598.

whereas the classical algorithm is $O(n^3)$. The principal disadvantage is that it is not stable for certain row or column scalings [4]. A minor nuisance is that it also requires extra storage for saving some of the intermediate matrix calculations (approximately $2n^2/3$).

The operation counts do not give a complete feel for the value of using (1) instead of the classical algorithm. For extremely large matrices, the multiplications will recursively be done using (1) until the matrices are small enough so that the classical algorithm is more efficient to use. Each level of recursion saves some time. Hence, (1) is only marginally superior to the classical algorithm until several levels of recursion take place.

2 Matmul Extension

The normal syntax for the Fortran–90 intrinsic function *matmul* is

$$C = matmul(A, B)$$

where A, B, and C are matrices of the proper dimensions.

The extension to Fortran–90 that has been implemented by IBM first in its XLF 3.1.1 compiler is to add an optional third argument, *mindim*:

$$C = matmul(A, B, mindim)$$

Mindim is an integer that determines whether or not to do the matrix multiplication using the classical algorithm. The dimensions of matrices are recursively split according to the rules in [3] until one of the dimensions is less than *mindim*. Then the classical algorithm is used to do the multiplication.

On an IBM RISC System/6000™ model 560 with 128 megabytes of memory, reasonable values for *mindim* are the following:

Data type	*mindim*
real*4	94
real*8	190
complex*8	25
complex*16	25
integer*4	51

However, the exact crossover point varies according to many factors, including the machine configuration, available memory, and the type of data in the matrices. Real, complex, and integer matrices of various data precisions can use the new functionality.

Setting $mindim = 1$ is reserved for future use. The obvious use is for the intrinsic function to choose a good value for *mindim* for the user. When $mindim \leq 0$, the classical algorithm is used.

We computed speedups for single precision real matrices using $mindim = 94$. Square matrices of order n were multiplied together. The elements of A and B were chosen at random using the Fortran–90 intrinsic function *random_number*. The elements lay in the range $(0, 1)$. Typical speedups over the classical algorithm are the following:

n	Speedup
200	1.0270
300	1.0957
400	1.1862
500	1.2082
600	1.2099
700	1.2632
800	1.3241
900	1.3211
1000	1.3437
1100	1.3648
1200	1.3883
1300	1.3943
1400	1.4271

A highly portable implementation of (1) is available on netlib (*gemmw* in the linalg directory; see [3]). This is not the IBM implementation (which is written in Fortran–90). *Gemmw* is written in a combination of C, Fortran, and calls to commonly available libraries (e.g., BLAS). Common values for *mindim* in *gemmw* are 32, 64, 96, 128, 192, and 256. While the optimal value of *mindim* for a given machine may not be one of these values, a nearly optimal value has been determined based on these six values for every machine tested to date.

3 Conclusions

Thanks in part to the flexibility of Fortran–90, fast algorithms can be conveniently added to the run time library of any Fortran–90 compiler. For matrix–matrix multiplication, there is already one such extension in existence today. Based on the highly portable *gemmw* code on netlib, adding this to other Fortran–90's should be both easy and straight forward.

References

[1] ANSI. *American National Standard Programming Language Fortran 90.* New York, X3.198–1992 edition, 1992.

[2] C. C. Douglas and J. Douglas. A unified convergence theory for abstract multigrid or multilevel algorithms, serial and parallel. *SIAM J. Numer. Anal.*, 30:136–158, 1993.

[3] C. C. Douglas, M. Heroux, G. Slishman, and R. M. Smith. Gemmw: A portable Level 3 BLAS Winograd variant of Strassen's matrix–matrix multiply algorithm. *J. Comput. Phys.*, 110:1–10, 1994.

[4] N. J. Higham. Exploiting fast matrix multiplication within the Level 3 BLAS. *ACM Trans. Math. Soft.*, 16:352–368, 1990.

[5] IBM. *AIX XL Fortran Compiler/6000: Language Reference.* North York, Ontario, 3, release 2 edition, 1994.

[6] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[7] S. Winograd. Some remarks on fast multiplication of polynomials. In J. F. Traub, editor, *Complexity of Sequential and Parallel Numerical Algorithms*, pages 181–196. Academic Press, New York, 1973.