An Error-Free Transformation for Matrix Multiplication with Reproducible Algorithms and Divide and Conquer Methods

Katsuhisa Ozaki
307 Fukasaku, Minuma-ku, Saitamashi, Saitama 337-8570, Japan
E-mail: ozaki@sic.shibaura-it.ac.jp

Abstract. This paper discusses accurate numerical algorithms for matrix multiplication. Matrix multiplication is a basic and important problem in numerical linear algebra. Numerical computations using floating-point arithmetic can be quickly performed on existing computers. However, the accumulation of rounding errors due to finite precision arithmetic is a critical problem. An error-free transformation for matrix multiplication is reviewed in this paper. Such a transformation is extremely useful for developing accurate numerical algorithms for matrix multiplication. One advantage of the transformation is that it exploits Basic Linear Algebra Subprograms (BLAS). We provide a rounding error analysis of reproducible algorithms for matrix multiplication based on the error-free transformation. In addition, we propose an error-free transformation for matrix multiplication that can be utilized with the divide and conquer methods.

1. Introduction
Matrix multiplication is a basic and important problem in numerical linear algebra. Let $F$ be a set of binary floating-point numbers as defined in IEEE 754 [1]. Let $u$ be unit roundoff, e.g., $u = 2^{-53}$ for binary64 in IEEE 754. For $A \in F^{m \times n}$ and $B \in F^{n \times p}$, let $\hat{C}$ denote a computed result of $AB$ via floating-point arithmetic in the IEEE 754 format. Then, from [2], we have

$$|\hat{C} - AB| \leq nu|A||B|$$

barring overflow and underflow in the evaluation of the matrix multiplication. This provides the upper bound of the relative error

$$\frac{|\hat{c}_{ij} - (AB)_{ij}|}{|(AB)_{ij}|} \leq nu \frac{|A||B|_{ij}}{|(AB)_{ij}|}, \quad (AB)_{ij} \neq 0. \quad (2)$$

If heavy cancellation occurs in $(AB)_{ij}$ or $n$ is large, the relative error may become substantial. Multi-precision arithmetic is one potential solution to overcome this problem because $u$ in (1) and (2) becomes very small. However,

- the computational cost for multi-precision arithmetic is expensive compared to hardware supported floating-point arithmetic and
all problems are not ill-conditioned. Multi-precision arithmetic is not necessary for well-conditioned problems, and it is initially unknown whether a problem is well- or ill-conditioned.

We can apply an accurate and adaptive algorithm for summation of floating-point numbers [3, 4, 5] to matrix multiplication. For \( a, b \in \mathbb{F} \), \( ab \) can be transformed into \( x + y \), \( x, y \in \mathbb{F} \) using an error-free transformation [6]. Therefore, a dot product of \( n \)-vectors can be transformed into a sum of \( 2n \) floating-point numbers. A matrix multiplication consists of dot products; therefore, accurate algorithms for summation can be applied to matrix multiplication. However, the matrix multiplication performance is extremely high due to optimization techniques for acceleration. For example, there are optimized Basic Linear Algebra Subprograms (BLAS) such as the Intel Math Kernel Library and Open BLAS. Routines in BLAS are well parallelized, and the performance of these routines is nearly at its peak. Applying accurate algorithms for summation in an element-wise manner does not provide as high a performance as matrix multiplication.

An error-free transformation for matrix multiplication was proposed [7] wherein matrices \( A \) and \( B \) can be divided into

\[
A = A^{(1)} + \cdots + A^{(k)}, \quad A^{(i)} \in \mathbb{F}^{m \times n}, \quad 1 \leq i \leq k; \\
B = B^{(1)} + \cdots + B^{(\ell)}, \quad B^{(j)} \in \mathbb{F}^{n \times p}, \quad 1 \leq j \leq \ell.
\]

We can compute \( A^{(i)}B^{(j)} \) without a rounding error for all \((i, j)\) pairs. Then, \( AB \) can be transformed into an unevaluated sum of \( k\ell \) floating-point matrices. After this transformation, we apply the accurate summation algorithms. If elements in \( A \) and \( B \) are randomly generated from a pseudo normal distribution, \( k \) and \( \ell \) are 3, 4, or 5 in many cases. Therefore, the main computational cost herein is the matrix multiplication.

2. An Error-Free Transformation for Matrix Multiplication

We review an error-free transformation for matrix multiplication \( AB \) for \( A = (a_{ij}) \in \mathbb{F}^{m \times n} \) and \( B = (b_{ij}) \in \mathbb{F}^{n \times p} \) [7]. Other variants of error-free transformations for matrix multiplication are also provided [8, 9]. We assume \( n \ll u^{-1} \). First, we define a constant \( \beta \) such that

\[
\beta = \left\lceil \frac{\log_2 n - \log_2 u}{2} \right\rceil. \tag{3}
\]

We also define two vectors \( \sigma^{(1)} \in \mathbb{F}^m, \tau^{(1)} \in \mathbb{F}^p \) as

\[
\sigma_i^{(1)} = 2^\beta \cdot 2^x_i^{(1)}, \quad \tau_j^{(1)} = 2^\beta \cdot 2^y_j^{(1)},
\]

where the two vectors \( x^{(1)} \) and \( y^{(1)} \) are defined by

\[
x_i^{(1)} = \lceil \log_2 \max_{1 \leq j \leq n} |a_{ij}| \rceil, \quad y_j^{(1)} = \lceil \log_2 \max_{1 \leq i \leq n} |b_{ij}| \rceil. \tag{4}
\]

We define an exception for (4): \( x_i^{(1)} = 0 \) if \( \max_{1 \leq j \leq n} |a_{ij}| \), \( y_j^{(1)} = 0 \) if \( \max_{1 \leq i \leq n} |b_{ij}| \). We obtain \( a_{ij}^{(1)}, a_{ij}^{(2)} \), \( b_{ij}^{(1)} \), and \( b_{ij}^{(2)} \) as follows:

\[
a_{ij}^{(1)} = \text{fl}(a_{ij} + \sigma_i^{(1)}) - \sigma_i^{(1)}, \quad a_{ij}^{(2)} = \text{fl}(a_{ij} - a_{ij}^{(1)}), \\
b_{ij}^{(1)} = \text{fl}(b_{ij} + \tau_j^{(1)}) - \tau_j^{(1)}, \quad b_{ij}^{(2)} = \text{fl}(b_{ij} - b_{ij}^{(1)}). \tag{5}
\]
Then, \( A = A^{(1)} + A^{(2)} \), \( B = B^{(1)} + B^{(2)} \) is satisfied. Next, we define \( \sigma^{(2)} \) and \( \tau^{(2)} \) from \( A^{(2)} \) and \( B^{(2)} \) using

\[
\sigma_i^{(2)} = 2^\beta \cdot 2^{x_i^{(2)}}, \quad \tau_j^{(2)} = 2^\beta \cdot 2^{y_j^{(2)}},
\]

where \( x^{(2)} \) and \( y^{(2)} \) are defined using

\[
x_i^{(2)} = \lceil \log_2 \max_{1 \leq j \leq n} |a_{ij}^{(2)}| \rceil, \quad y_j^{(2)} = \lceil \log_2 \max_{1 \leq i \leq n} |b_{ij}^{(2)}| \rceil.
\]

Similarly we define an exception for (4): \( x_i^{(2)} = 0 \) if \( \max_{1 \leq j \leq n} |a_{ij}^{(2)}| \), \( y_j^{(2)} = 0 \) if \( \max_{1 \leq i \leq n} |b_{ij}^{(2)}| \). Using these vectors, we can compute

\[
a_{ij}^{(2)} = \text{fl}(a_{ij}^{(2)} + \sigma_i^{(2)}) - \sigma_i^{(2)}, \quad a_{ij}^{(3)} = \text{fl}(a_{ij}^{(2)} - a_{ij}^{(2)}),
\]

\[
b_{ij}^{(2)} = \text{fl}(b_{ij}^{(2)} + \tau_j^{(2)}) - \tau_j^{(2)}, \quad b_{ij}^{(3)} = \text{fl}(b_{ij}^{(2)} - b_{ij}^{(2)})
\]

for all \((i, j)\) elements. Here,

\[
A^{(2)} = A^{(2)} + A^{(3)}, \quad A = A^{(1)} + A^{(2)} + A^{(3)},
\]

\[
B^{(2)} = B^{(2)} + B^{(3)}, \quad B = B^{(1)} + B^{(2)} + B^{(3)}
\]

are satisfied. Let \( A^{(1)} = A \) and \( B^{(1)} = B \). Generally, we set \( \sigma^{(w)} \) and \( \tau^{(w)} \) to

\[
\sigma_i^{(w)} = 2^\beta \cdot 2^{x_i^{(w)}}, \quad \tau_j^{(w)} = 2^\beta \cdot 2^{y_j^{(w)}},
\]

(6) where \( x^{(w)} \) and \( y^{(w)} \) are defined using

\[
x_i^{(w)} = \lceil \log_2 \max_{1 \leq j \leq n} |a_{ij}^{(w)}| \rceil, \quad y_j^{(w)} = \lceil \log_2 \max_{1 \leq i \leq n} |b_{ij}^{(w)}| \rceil,
\]

(7) with exception if \( \max_{1 \leq j \leq n} |a_{ij}^{(w)}| \) and \( y_j^{(w)} = 0 \) if \( \max_{1 \leq i \leq n} |b_{ij}^{(w)}| \). We obtain \( A^{(w)}, B^{(w)}, A^{(w+1)} \) and \( B^{(w+1)} \) using

\[
a_{ij}^{(w)} = \text{fl}(a_{ij}^{(w)} + \sigma_i^{(w)}) - \sigma_i^{(w)}, \quad a_{ij}^{(w+1)} = \text{fl}(a_{ij}^{(w)} - a_{ij}^{(w)}),
\]

\[
b_{ij}^{(w)} = \text{fl}(b_{ij}^{(w)} + \tau_j^{(w)}) - \tau_j^{(w)}, \quad b_{ij}^{(w+1)} = \text{fl}(b_{ij}^{(w)} - b_{ij}^{(w)})
\]

(8)

We compute (6) and (8) for \( w = 1, 2, \ldots, \) there exist constants \( n_A, n_B \in \mathbb{N} \) such that

\[
A = \sum_{r=1}^{n_A} A^{(r)}, \quad B = \sum_{s=1}^{n_B} B^{(s)}, \quad A^{(n_A+1)} = O_{nn}, \quad B^{(n_B+1)} = O_{nn},
\]

(9)

where \( O_{mn} \) denotes the \( m \times n \) zero matrix. Therefore, the matrix product \( AB \) can be calculated as

\[
AB = \left( \sum_{r=1}^{n_A} A^{(r)} \right) \left( \sum_{s=1}^{n_B} B^{(s)} \right).
\]

(10)

Assume that a routine for matrix multiplication computes dot product. The dot product involves \( n \) products and \( n-1 \) additions. For \( n-1 \) additions, any order of the computations such as recursive, blockwise or pairwise orders are accepted. However, assume that the divide and conquer methods, e.g. [10, 11], are not applied to the matrix multiplication. Then, it has been proved that no rounding error occurs in \( \text{fl}(A^{(r)}B^{(s)}) \), \( 1 \leq r \leq n_A, \ 1 \leq s \leq n_B \) [7]. Therefore, using (10), \( AB \) can be transformed into an unevaluated sum of \( n_A n_B \) floating-point matrices.
3. Rounding Error Analysis for Reproducible Algorithms

Reproducibility is important in general sciences, and recently there have been frequent discussions concerning the reproducibility of numerical computations. A reproducible algorithm in numerical computations produces always bitwise identical results from multiple runs of a program on the same input [12]. There are libraries of reproducible BLAS [13, 14]. We introduce reproducible numerical algorithms for matrix multiplication based on the error-free transformation [15]. Assume that $A$ and $B$ are divided into $k$ floating-point matrices using (8) as follows

$$A = A^{(1)} + \cdots + A^{(k-1)} + A^{(k)}, \quad B = B^{(1)} + \cdots + B^{(k-1)} + B^{(k)}.$$  

Because $A^{(i)} B^{(j)} = \text{fl}(A^{(i)} B^{(j)})$ for $i + j \leq k$, we have

$$AB = \sum_{i+j \leq k} A^{(i)} B^{(j)} + \sum_{i=1}^{k-1} A^{(i)} B^{(k-i+1)} + A^{(k)} B$$

$$= \sum_{i+j \leq k} \text{fl}(A^{(i)} B^{(j)}) + \sum_{i=1}^{k-1} A^{(i)} B^{(k-i+1)} + A^{(k)} B. \quad (11)$$

If we partially use the terms in (11),

$$AB \approx \sum_{i+j \leq k} \text{fl}(A^{(i)} B^{(j)}).$$

For the sum $\sum_{i+j \leq k} \text{fl}(A^{(i)} B^{(j)})$, we apply the NearSum algorithm (Algorithm 7.4 [5]), which produces the nearest floating-point number to the exact result. Then, we can obtain the reproducible result, because there is no rounding error in matrix multiplication for any order of computations.

For the computed result $\hat{C}_k \in \mathbb{F}^{m \times p}$, we can provide an error bound for $\hat{C}_k$ from the result computed by the NearSum algorithm:

$$| \sum_{i+j \leq k} \text{fl}(A^{(i)} B^{(j)}) - \hat{C}_k | \leq u |\hat{C}_k|. \quad (12)$$

From (11), we obtain

$$|AB - \sum_{i+j \leq k} \text{fl}(A^{(i)} B^{(j)})| \leq \sum_{i=1}^{k-1} |A^{(i)} B^{(k-i+1)} + A^{(k)} B|$$

$$\leq \sum_{i=1}^{k-1} |A^{(i)}| |B^{(k-i+1)}| + |A^{(k)}||B| \quad (13)$$

From (12) and (13), we have

$$|AB - \hat{C}_k| \leq u |\hat{C}_k| + \sum_{i=1}^{k-1} |A^{(i)}| |B^{(k-i+1)}| + |A^{(k)}||B|$$

From (6) and (8), we derive

$$|\epsilon^{(k)}_{ij}|, |\delta^{(k)}_{ij}| \leq (2^\beta u)^{k-1} \cdot x^{(1)}_i, \quad |\epsilon^{(k)}_{ij}| \leq (2^\beta u)^{k-1} \cdot y^{(1)}_j,$$
and we obtain

\[ |AB - \hat{C}_k| \leq u|\hat{C}_k| + nk(2^\beta u)^{k-1} \cdot x^{(1)}(y^{(1)})^T. \]  

(14)

Let \( E_A \in \mathbb{F}^{m \times n} \) and \( E_B \in \mathbb{F}^{n \times p} \) (all elements of \( E_A \) and \( E_B \) are ones). If \( |A| \approx (\max_{i,j} a_{ij})E_A \) and \( |B| \approx (\max_{i,j} b_{ij})E_B \), then \( nx^{(1)}(y^{(1)})^T \approx |A||B| \). From this and (14), we have

\[ |AB - \hat{C}_k| \lesssim k(2^\beta u)^{k-1}|A||B|. \]  

(15)

Therefore, if \( n \leq 100,000 \) and \( k = 3 \), the bound (15) is comparable to (1).

4. Extension to Divide and Conquer Methods

In Section 2, we assumed that the divide and conquer methods were not applied to the matrix multiplication routine. Here, we adopt the error-free transformation to the matrix multiplication computed by the divide and conquer methods.

For \( A \in \mathbb{F}^{m \times n} \) and \( B \in \mathbb{F}^{n \times p} \), assume that \( m, n \) and \( p \) are even. We consider the following block division of \( C = AB \).

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
= \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}.
\]

We introduce the Strassen’s algorithm for matrix multiplication [10]. Matrices \( P_i, i = 1, 2, \ldots, 7 \) are defined as follows:

\[
P_1 = (A_{11} + A_{22})(B_{11} + B_{22}), \quad P_2 = (A_{21} + A_{22})B_{11},
\]

\[
P_3 = A_{11}(B_{12} - B_{22}), \quad P_4 = A_{22}(B_{21} - B_{11}), \quad P_5 = (A_{11} + A_{12})B_{22}, \quad P_6 = (A_{21} - A_{11})(B_{11} + B_{12}), \quad P_7 = (A_{12} - A_{22})(B_{21} + B_{22}).
\]

(16)

Then, each block in \( C \) can be obtained by

\[
C_{11} = P_1 + P_4 - P_5 + P_7, \quad C_{12} = P_3 + P_5, \quad C_{21} = P_2 + P_4, \quad C_{22} = P_1 + P_3 - P_2 + P_6.
\]

Next, we introduce the Winograd’s method for matrix multiplication [11]. We set eight matrices as follows.

\[
S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2, \\
S_5 = B_{12} - B_{11}, \quad S_6 = B_{22} - B_{12}, \quad S_7 = B_{22} - B_{12}, \quad S_8 = S_6 - B_{21}.
\]

We compute seven matrices from \( M_1 \) to \( M_7 \) as follows:

\[
M_1 = S_2S_6, \quad M_2 = A_{11}B_{11}, \quad M_3 = A_{12}B_{21}, \quad M_4 = S_3S_7, \\
M_5 = S_1S_5, \quad M_6 = S_4B_{22}, \quad M_7 = A_{22}S_8.
\]

(17)

Let two matrices \( T_1 \) and \( T_2 \) be

\[
T_1 = M_1 + M_2, \quad T_2 = T_1 + M_4.
\]

Finally, the matrix \( C \) is obtained by

\[
C_{11} = M_2 + M_3, \quad C_{12} = T_1 + M_5 + M_6, \quad C_{21} = T_2 - M_7, \quad C_{22} = T_2 + M_5.
\]

We can apply the Strassen (or Winograd) algorithm again for the products of the matrices in (16) and (17).
Assume that the matrix size $m$, $n$ and $p$ are multiples of $2^k$, $k \in \mathbb{Z}$, and the Strassen or Winograd method is applied $k$ times recursively for a routine computing the matrix multiplication. We modify the constant $\beta$ in (3) for the divide and conquer methods such that
\[
\beta = \left\lceil \frac{\log_2 n/2^k - \log_2 u}{2} \right\rceil
\] (18)
and vectors $\sigma^{(1)}$ and $\tau^{(1)}$ as
\[
\sigma_i^{(1)} = 2 \cdot 2^k \cdot 2^\beta \cdot 2^{\sigma_i^{(1)}} , \quad \tau_j^{(1)} = 2 \cdot 2^k \cdot 2^\beta \cdot 2^{\tau_j^{(1)}},
\]
where two vectors $x^{(1)}$ and $y^{(1)}$ are defined by
\[
x_i^{(1)} = [\log_2 \max_{1 \leq i, j \leq n} |a_{ij}|] , \quad y_j^{(1)} = [\log_2 \max_{1 \leq i, j \leq n} |b_{ij}|].
\] (19)
We obtain $a_{ij}^{(1)}$, $\underline{a}_{ij}^{(2)}$, $b_{ij}^{(1)}$ and $\underline{b}_{ij}^{(2)}$ as follows:
\[
a_{ij}^{(1)} = \lceil (a_{ij} + \sigma_i^{(1)}) - \sigma_i^{(1)} \rceil , \quad \underline{a}_{ij}^{(2)} = \lceil (a_{ij} - \sigma_i^{(1)}) \rceil
\]
\[
b_{ij}^{(1)} = \lceil (b_{ij} + \tau_j^{(1)}) - \tau_j^{(1)} \rceil , \quad \underline{b}_{ij}^{(2)} = \lceil (b_{ij} - \tau_j^{(1)}) \rceil.
\] (20)
Then,
\[
A = A^{(1)} + A^{(2)}, \quad B = B^{(1)} + B^{(2)}.
\]
Generally, assume $\sigma^{(w)}$, $\tau^{(w)}$ to be
\[
\sigma_i^{(w)} = 2 \cdot 2^k \cdot 2^\beta \cdot 2^{\sigma_i^{(w)}}, \quad \tau_j^{(w)} = 2 \cdot 2^k \cdot 2^\beta \cdot 2^{\tau_j^{(w)}},
\] (21)
where $x^{(w)}$ and $y^{(w)}$ are defined by
\[
x_i^{(w)} = [\log_2 \max_{1 \leq i, j \leq n} |a_{ij}^{(w)}|] , \quad y_j^{(w)} = [\log_2 \max_{1 \leq i, j \leq n} |b_{ij}^{(w)}|].
\]
We obtain $a_{ij}^{(w)}$, $\underline{a}_{ij}^{(w+1)}$, $b_{ij}^{(w)}$ and $\underline{b}_{ij}^{(w+1)}$ by
\[
a_{ij}^{(w)} = \lceil (a_{ij}^{(w)} + \sigma_i^{(w)}) - \sigma_i^{(w)} \rceil , \quad \underline{a}_{ij}^{(w+1)} = \lceil (a_{ij}^{(w)} - a_{ij}^{(w)}) \rceil
\]
\[
b_{ij}^{(w)} = \lceil (b_{ij}^{(w)} + \tau_j^{(w)}) - \tau_j^{(w)} \rceil , \quad \underline{b}_{ij}^{(w+1)} = \lceil (b_{ij}^{(w)} - b_{ij}^{(w)}) \rceil.
\] (22)
Computing (21) and (22) for $w = 1, 2, \ldots$, there exist some constants $n'_A, n'_B \in \mathbb{N}$ such that
\[
A = \sum_{r=1}^{n'_A} A^{(r)} , \quad B = \sum_{s=1}^{n'_B} B^{(s)} , \quad A^{(n'_A+1)} = O_{mn}, \quad B^{(n'_B+1)} = O_{np}.
\] (23)
Then,
\[
A^{(i)} B^{(j)} = \lceil A^{(i)} B^{(j)} \rceil , \quad 1 \leq i \leq n'_A, \quad 1 \leq j \leq n'_B
\] (24)
is satisfied for all $(i, j)$ pairs. Even if the divide and conquer methods are not applied to the computing routine for matrix multiplication, (23) and (24) will still be valid.
Conclusions
We have reviewed an error-free transformation for matrix multiplication and the upper bound of
the absolute error for reproducible algorithms based on the error-free transformation has been
provided. In addition, we have adjusted the constants for the error-free transformation to apply
it to the divide and conquer methods.

References
[1] IEEE Standard for Floating-Point Arithmetic, Std 754–2008 2008
[2] Jeannerod, C -P and Rump S M 2013 SIAM. J. Matrix Anal. & Appl. 34 338
[3] Ogita T, Rump S M and Oishi S 2005 SIAM J. Sci. Comput. 26 1955
[4] Rump S M, Ogita T and Oishi S 2008 SIAM J. Sci. Comput. 31 189
[5] Rump S M, Ogita T and Oishi S 2008 SIAM J. Sci. Comput. 31 1269
[6] Dekker T J 1971 Numer. Math. 18 224
[7] Ozaki K, Ogita T, Oishi S and Rump S M 2012 Numer. Algorithms 59 95
[8] Ozaki K, Ogita T, Oishi S and Rump S M 2013 NOLTA, IEICE 4 2
[9] Ozaki K, Ogita T and Oishi S 2016 Numer. Lin. Algebra Appl. 23 931
[10] Strassen V 1969 Numer. Math. 13 354
[11] Winograd S 1971 Linear Algebra Its Appl. 4 381
[12] Ahrens, P, Nguyen H D and Demmel J 2016 EECS Department, UC Berkeley, Technical Report No.
UCB/EECS-2016-121,
[13] ReproBLAS: Reproducible BLAS, https://bebop.cs.berkeley.edu/reproblas/
[14] ExBLAS – Exact BLAS, https://exblas.lip6.fr/index.php
[15] Mukunoki D and Ogita T 2019 private communication