2008 International Symposium on Nonlinear Theory and its Applications
NOLTA'08, Budapest, Hungary, September 7-10, 2008

Accurate Matrix Multiplication by using Level 3 BLAS Operation

Katsuhisa Ozaki†, Takeshi Ogita‡,†, Siegfried M. Rump* and Shin'ichi Oishi†,**

†Faculty of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjyuku-ku, Tokyo, 169-0075 Japan
‡Department of Mathematics, Tokyo Woman's Christian University
2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan
* Institute for Reliable Computing, Hamburg University of Technology
Schwarzenbergstr. 95, 21071 Hamburg, Germany
** CREST, Japan Science and Technology Agency
Email: k_ozaki@aoni.waseda.jp

Abstract—This paper is concerned with an accurate computation of matrix multiplication. Recently, an accurate summation algorithm was developed by Rump, Ogita and Oishi. One of the key techniques of their method is a new type of error-free splitting. To use this strategy, we investigate a method of obtaining an accurate result of matrix multiplication by mainly using Level 3 BLAS operation. Finally, we present numerical examples showing the effectiveness of the proposal algorithm.

1. Introduction

This paper is concerned with an accurate computation for matrix multiplication. Let A and B be floating-point matrices. To obtain an accurate result of matrix multiplication AB, there are the following possibilities:

- Accurate sum and dot product [1, 6]
- Multiple-precision arithmetic [5, 7]

Recently, an accurate summation algorithm was developed by Rump, Ogita and Oishi [3, 4], which outputs a faithfully rounded result of the exact sum of floating-point numbers. This method avoids using a sorting for input data, branch in the main loops and direct access for mantissa bit or exponent bit which make a performance of computation slow. Therefore, their method is not only less amount of computations but also fast in terms of measured computing time. Moreover, only usual floating-point arithmetic is needed in their algorithm so that we can portably implement their algorithm.

In this paper, we investigate a method of computing accurate matrix multiplication by using mainly level 3 BLAS (Basic Linear Algebra Subroutines). Based on the strategy of the accurate summation algorithm [3], the proposed method does not need multi-precision library nor some extended precision arithmetic like double-extended precision. Since BLAS routines are optimized for an architecture in use, such an optimized BLAS is so fast that some BLAS routines achieve near peak performance. Moreover, most of the optimized BLAS automatically adapts to multithreaded environments. Our method mainly uses this BLAS, therefore, it is easy to accommodate to parallel computation without changing a serial code.

Finally, we present numerical examples for illustrating the effectiveness of proposed method.

2. Error-Free Split

In this section, we define notations used in this paper and explain the key technique of the accurate summation algorithm `AccSum` developed by Rump, Ogita and Oishi. All computations are done by floating-point arithmetic defined by IEEE 754, especially, double precision. Let $\mathbb{F}$ be a set of floating-point number and $\mathbf{u} = 2^{-53}$ be unit roundoff. $fl(\cdots)$ shows that an expression inside the parenthesis is computed by floating-point arithmetic. We use a Matlab code for readability.

For a floating-point number $p \in \mathbb{F}$, let σ and M be constants defined by

$$\sigma = 2^M 2^{\lceil \log_2 |p| \rceil},\ M = 2^k\ (k \in \mathbb{Z}).$$

The following algorithm plays an important role in `AccSum`.

Algorithm 1 (Rump-Ogita-Oishi [3]) *For $p \in \mathbb{F}$, this algorithm transforms $q + p' = p$.*

```
function [q, p'] = ExtractScalar(p, σ)
  q = fl((σ + p) − σ);
  p' = fl(p − q);
```

Here, we introduce the properties of q and p'. We assume that

$$|p| < 2^{-M}\sigma \tag{1}$$

is satisfied. When executing

$$[q, p'] = \texttt{ExtractScalar}(p, \sigma),$$

it holds from [3] that

$$|p'| \leq \mathbf{u}\sigma \tag{2}$$
$$q \leq \sigma 2^{-M} \tag{3}$$
$$q \in \mathbf{u}\sigma\mathbb{Z} \tag{4}$$

We use the error-free splitting and the properties (2)–(4) to develop a method obtaining the accurate result of matrix multiplication.

3. Improvement of accuracy for matrix multiplication

In this section, we specialize the strategy of the accurate summation algorithm and investigate a method of improving the accuracy of matrix multiplication.

First, we define a constant M' as

$$M' = \left\lceil \frac{\log_2(n+1)+53}{2} \right\rceil. \tag{5}$$

Let A and B be $A \in \mathbb{F}^{m\times n}, B \in \mathbb{F}^{n\times p}$ respectively. Let m_a and m_b be

$$m_a = \max_{1\le i\le m, 1\le j\le n} |A_{ij}|,\ m_b = \max_{1\le i\le n, 1\le j\le p} |B_{ij}|.$$

Next we define two constants σ', σ'' as

$$\sigma' = 2^{M'}2^{\lceil \log_2 m_a\rceil},\ \sigma'' = 2^{M'}2^{\lceil \log_2 m_b\rceil}. \tag{6}$$

We here split A into two floating-point matrices satisfying

$$A = A^{(1)} + A^{(2)}. \tag{7}$$

We denote a matrix $E_A \in \mathbb{F}^{m\times n}$ whose all elements are ones. Using Algorithm 1, we have

$$A^{(1)} = (A + \sigma' E_A) - \sigma' E_A,\ A^{(2)} = A - A^{(1)}. \tag{8}$$

For B, we can similarly have

$$B^{(1)} = (B + \sigma'' E_B) - \sigma'' E_B,\ B^{(2)} = B - B^{(1)} \tag{9}$$

where $E_B \in \mathbb{F}^{n\times p}$ whose all elements are ones. After these splittings, matrix multiplication AB can be transformed as follows:

$$\begin{aligned} AB &= (A^{(1)} + A^{(2)})(B^{(1)} + B^{(2)}) \\ &= A^{(1)}B^{(1)} + A^{(1)}B^{(2)} + A^{(2)}B^{(1)} + A^{(2)}B^{(2)} \\ &= A^{(1)}B^{(1)} + A^{(1)}B^{(2)} + A^{(2)}B. \end{aligned} \tag{10}$$

Now we notice that there is no rounding error in the computation of $A^{(1)}B^{(1)}$.

Theorem 1 *From the computations (5), (6), (8) and (9), there is no rounding error in $fl(A^{(1)}B^{(1)})$ no matter what the order of floating-point operations so that*

$$fl(A^{(1)}B^{(1)}) = A^{(1)}B^{(1)}.$$

Proof. For (i, j) element of $A^{(1)}B^{(1)}$, we aim to prove

$$fl(\sum_{k=1}^{n} A^{(1)}_{ik}B^{(1)}_{kj}) = \sum_{k=1}^{n} A^{(1)}_{ik}B^{(1)}_{kj}.$$

From the definition of M' and a little consideration, we obtain

$$m_a \le 2^{-M'}\sigma',\ m_b \le 2^{-M'}\sigma''.$$

It shows that the assumption (1) is satisfied for all elements A and B, and we can use the relations (2)–(4). From (4), it holds that

$$A^{(1)}_{ik} \in \mathbf{u}\sigma'\mathbb{Z},\ B^{(1)}_{kj} \in \mathbf{u}\sigma''\mathbb{Z}. \tag{11}$$

From (11), we have

$$A^{(1)}_{ik}B^{(1)}_{kj} \in \mathbf{u}^2\sigma'\sigma''\mathbb{Z}$$

and

$$\sum_{k=1}^{n} A^{(1)}_{ik}B^{(1)}_{kj} \in \mathbf{u}^2\sigma'\sigma''\mathbb{Z}. \tag{12}$$

From (3), we obtain

$$A^{(1)}_{ik} \le 2^{-M'}\sigma',\ B^{(1)}_{kj} \le 2^{-M'}\sigma''.$$

Here we consider an upper bound of $|\sum A^{(1)}B^{(1)}|$:

$$\begin{aligned} \left|\sum_{k=1}^{n} A^{(1)}_{ik}B^{(1)}_{kj}\right| &\le \sum_{k=1}^{n} |A^{(1)}_{ik}||B^{(1)}_{kj}| \\ &\le n2^{-M'}\sigma'2^{-M'}\sigma'' \\ &= n2^{-2M'}\sigma'\sigma'' =: t \end{aligned} \tag{13}$$

To substitute (5) into (13), we obtain

$$\begin{aligned} t &\le n2^{-2M'}\sigma'\sigma'' = n2^{-2\lceil \frac{\log_2(n+1)+53}{2}\rceil}\sigma'\sigma'' \\ &= \frac{n\sigma'\sigma''}{2^{2\lceil \frac{\log_2(n+1)+53}{2}\rceil}} \le \frac{n\sigma'\sigma''}{2^{\log_2(n+1)+53}} \\ &= \frac{n}{(n+1)}\mathbf{u}\sigma'\sigma'' < \mathbf{u}\sigma'\sigma'' \end{aligned}$$

From (12) and (14), it is shown that the least bit of the result and all intermediate values in the dot product are multiple of $\mathbf{u}^2\sigma'\sigma''$ and the upper bound of the dot product is less than $\mathbf{u}\sigma'\sigma''$. Thus, there is no rounding error since all bits of the elements of $A^{(1)}B^{(1)}$ are inside of range of mantissa. These complete the proof. □

In (10), there may be rounding errors in the computation of $A^{(1)}B^{(2)}$ and $A^{(1)}B$. This is an a priori error analysis for matrix multiplication:

$$|AB - fl(AB)| \le \gamma_n|A||B|,\ \gamma_n = \frac{n\mathbf{u}}{1-n\mathbf{u}}$$

For our method, it holds

$$\begin{aligned} |A^{(1)}B^{(2)} - fl(A^{(1)}B^{(2)})| &\le \gamma_n|A^{(1)}||B^{(2)}|, \\ |A^{(2)}B - fl(A^{(2)}B)| &\le \gamma_n|A^{(2)}||B|. \end{aligned}$$

Here, the element of $A^{(2)}$ and $B^{(2)}$ seem to be smaller than those of $A^{(1)}$ and B, respectively. Therefore, it is expected that the accuracy of result will be improved.

To make an algorithm more efficient, we discuss to take σ' and σ'' smaller. Considering the independency of computations for matrix multiplication, we can independently take σ' row-wise and σ'' column-wise. We redefine σ' and σ'' as a floating-point vector such that

$$\sigma'_i = 2^{M'}2^{\log_2 \max_{1\le j\le p}|A_{ij}|},\ \sigma''_i = 2^{M'}2^{\log_2 \max_{1\le i\le r}|B_{ij}|}.$$

We define the following two matrices:

$$P_1 = \mathrm{diag}(\sigma'),\ P_2 = \mathrm{diag}(\sigma'')$$

To split A and B, we compute the following procedures:

$$A^{(1)} = (A + P_1E_A) - P_1E_A \quad , \quad A^{(2)} = A - A^{(1)}$$
$$B^{(1)} = (B + E_BP_2) - E_BP_2 \quad , \quad B^{(2)} = B - B^{(1)}$$

We here present an algorithm:

Algorithm 2 Let $A \in \mathbb{F}^{m\times n}, B \in \mathbb{F}^{n\times p}$, this algorithm computes matrix multiplication AB.

function C = `Mul2`(A, B)
  $mu = \max(\mathrm{abs}(A), [], 2)$;
  $temp = 2.\hat{}(\mathrm{ceil}(\log 2(mu) + (53 + \log 2(q))/2))$;
  $\sigma' = \mathrm{repmat}(temp, 1, q)$;
  $A^{(1)} = (A + \sigma') - \sigma'$;
  $A^{(2)} = A - A^{(1)}$;
  $mu = \max(\mathrm{abs}(B))$;
  $temp = 2.\hat{}(\mathrm{ceil}(\log 2(mu)) + \mathrm{ceil}((53 + \log 2(q))/2))$;
  $\sigma'' = \mathrm{repmat}(temp, q, 1)$;
  $B^{(1)} = (B + \sigma'') - \sigma''$;
  $B^{(2)} = B - B^{(1)}$;
  $C = A^{(1)}B^{(1)} + (A^{(1)}B^{(2)} + A^{(2)}B)$;
  end

Remark 1 *It is better to use an accurate summation algorithm to* $A^{(1)}B^{(1)} + (A^{(1)}B^{(2)} + A^{(2)}B)$ *in Algorithm 2. Error is affected from the order of computation. We recommend to use* Sum2 *[1] or* ExpansionSum *[2] and so on. However, total computational cost is almost not changed.*

Comparing to a result obtained by usual floating-point arithmetic, it is expected that the accuracy of a result from proposal method will be improved.

Our method needs matrix multiplication 3 times. The cost of error-free splittings is almost negligible when n is not so small compared to m and p.

If we split A and B into many matrices, it is expected that the accuracy of the computed result is improved more effectively. For example, we split the matrices into 3 parts by the similar procedure to (5),(6), (8) and (9):

$$A = \sum_{i=1}^{3} A^{(i)},\ B = \sum_{i=1}^{3} B^{(i)}$$

then we can compute AB as

$$\begin{aligned} AB &= A^{(1)}B^{(1)} + A^{(1)}B^{(2)} + A^{(2)}B^{(1)} \\ &+ A^{(1)}B^{(3)} + (A^{(2)} + A^{(3)})(B^{(2)} + B^{(3)}) + A^{(3)}B^{(1)} \end{aligned}$$

It involves 6 matrix products. In this computation, there is no rounding error in $fl(A^{(1)}B^{(1)}), fl(A^{(1)}B^{(2)})$ and $fl(A^{(2)}B^{(1)})$. When splitting A and B into k matrices, respectively,

$$A = \sum_{i=1}^{k} A^{(i)},\ B = \sum_{i=1}^{k} B^{(i)}.$$

We can prove the following result:

$$fl(A'_iB'_j) = A'_iB'_j,\ i + j \le k.$$

4. Reducing amount of working space

When we executing Algorithm 2, we must store $A_1, A_2, B_1, B_2, \sigma'(\sigma'')$ on a memory. We assume that a matrix is needed as working space in the computation of $C = A^{(1)}B^{(1)} + A^{(1)}B^{(2)} + A^{(2)}B$. If we apply Algorithm 2 straightforwardly, we must prepare the working space to store 6 matrices in total. It seems to be expensive so that we consider the way to reduce the amount of working space. For simplicity, we assume that A and B are n-by-n floating-point matrices, respectively.

Since all dot products in matrix multiplication are independent each other, we can divide C into many blocks and compute it independently. For example, when dividing C into 2-by-2 blocks, we compute the following four procedures:

$$C(1:r, 1:r) = \texttt{Mul2}(A(1:r, :), B(:, 1:r))$$
$$C(r+1:n, 1:r) = \texttt{Mul2}(A(r+1:n, :), B(:, 1:r))$$
$$C(1:r, r+1:n) = \texttt{Mul2}(A(1:r, :), B(:, r+1:n))$$
$$C(r+1:n, r+1:n) = \texttt{Mul2}(A(r+1:n, :), B(:, r+1:n))$$

where $r = \lceil n/2 \rceil$. Here we denote a constant s as the size of working space to store a matrix. Then, the amount of working space of $A_1, A_2, B_1, B_2, \sigma$ in Algorithm 2 become $s/2$ and the amount of working space storing an intermediate result in the computation C becomes $s/4$. Total working space becomes approximately

$$\frac{5}{2}s + \frac{1}{4}s = \frac{11}{4}s.$$

Generally, when we divide the matrix d-by-d parts, the required amount of working space becomes approximately

$$\frac{5}{d}s + \frac{1}{d^2}s$$

We present an algorithm saving the working space.

Algorithm 3 We apply `Mul2` in Algorithm 2 by using d-by-d block matrix products.

function C = `Mul2r`(A, B, d)
  $[m, n] = \mathrm{size}(A)$;
  $[n, p] = \mathrm{size}(B)$;
  $lm = 1 : \mathrm{ceil}(m/d) : m - 1$;
  $lp = 1 : \mathrm{ceil}(p/d) : p - 1$;
  $lm(d + 1) = m$;
  $lp(d + 1) = p$;
  for $i = 1 : d$
    for $j = 1 : d$
      $C(lm(i) : lm(i + 1), lp(j) : lp(j + 1)) =$
  `Mul2`$(A(lm(i) : lm(i + 1), :), B(:, lp(j) : lp(j + 1)))$;
    end
  end

It is certain that significant overhead occurs in Algorithm 3 on Matlab. We will check the performance of Algorithm 3 by numerical examples in the next section.

Table 1: Comparison of absolute errors.

ϕ	M1	M2
1	1.5916e-012	1.1369e-013
3	4.8828e-004	2.3842e-007
5	3.2768e+004	2.0480e+003
7	1.7592e+013	1.7180e+010
9	3.6893e+019	2.8823e+017

Table 2: Comparison of relative errors.

ϕ	M1	M2
1	5.1133e-008	3.5700e-012
3	9.5149e-008	5.6098e-011
5	2.1130e-008	3.0247e-009
7	3.4875e-009	2.2729e-009
9	2.3077e-009	3.6800e-009

5. Numerical Examples

In this section, we present numerical examples to illustrate the effectiveness of our algorithm. First, we generate the matrices A and B as

$$(\mathrm{rand}(10000) - 0.5). * \exp(\phi * \mathrm{randn}(10000)).$$

where the function 'rand' makes uniformly distributed pseudo random numbers. The function 'randn' returns a pseudo random, scalar value drawn from a normal distribution with mean 0 and standard deviation 1. Getting ϕ larger, there is big difference in the magnitude in the elements. We compare the accuracy of the following methods:

- M1: Usual floating-point computation ($2n^3$ flops)
- M2: Algorithm 2 ($6n^3$ flops)

Table 1 displays the maximum of absolute errors. Table 2 displays the maximum of relative errors. These results show that the accuracy is improved. When ϕ is large, the effectiveness of our method weakens since the constant σ' and σ'' are taken row-wise and column-wise respectively so that these constants are not suited for all dot products. Elapsed time of $M1$ is 48 sec and elapsed time of $M2$ is 155 sec on Core 2 Extreme 3.0 GHz and Matlab R2007b. It is certain that our method includes three matrix products so that elapsed time for M2 is about 3 times slower than that for M1.

Table 3 displays the elapsed times to execute

$$C = \mathtt{Mul2r}(A, B, d)$$

for various d. When d turns $d + 1$, the performance drops about 10 percent. It can be seen that there is trade off between the performance and required amount of memory.

Table 3: Comparison of elapsed time and amount of memory.

d	time	working space
1	155	6 s
2	173	11/4 s
3	191	16/9 s
4	206	13/8 s
5	216	26/25 s

In case of $d = 5$ from Table 3, we only prepare almost a matrix as working space.

The advantage of our method is that the optimized BLAS can be applied straightforwardly. Of course, we can improve the accuracy of dot product by using the loop. However, it slows down the performance on the interpreter language, for example, Matlab. Some routines from BLAS use multithreads automatically so that we can easily do a parallel computation.

Acknowledgments

This research was partially supported by CREST program, Japan Science and Technology Agency (JST) and Grant-in-Aid for Specially Promoted Research (No. 17002012: Establishment of Verified Numerical Computation) from the Ministry of Education, Science, Sports and Culture of Japan.

References

[1] T. Ogita, S. M. Rump, S. Oishi: Accurate sum and dot product, *SIAM J. Sci. Comput.*, 26 (2005), 1955–1988.

[2] J. R. Shewchuk: Adaptive precision floating-point arithmetic and fast robust geometric predicates, *Discrete & Computational Geometry* 18 (1997), 305–363.

[3] S. M. Rump, T. Ogita, S. Oishi: Accurate Floating-Point Summation Part I: Faithful Rounding, submitted for publication, 2007.

[4] S. M. Rump, T. Ogita, S. Oishi: Accurate Floating-Point Summation Part II: Sign, K-fold Faithful and Rounding to Nearest, submitted for publication, 2007.

[5] The MPFR Library: `http://www.mpfr.org/`

[6] J. Demmel, Y. Hida: Accurate and Efficient Floating Point Summation, SIAM J. Sci. Comput., 25(4):1214–1248, 2003.

[7] David H. Bailey: A Fortran-90 Based Multiprecision System, *ACM Transactions on Mathematical Software*, vol. 21, no. 4, 379-387.