2009 International Symposium on Nonlinear Theory and its Applications
NOLTA'09, Sapporo, Japan, October 18-21, 2009

Multiple Floating-point Matrix Multiplication by Level 3 Operations.

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 Mathematical Sciences, 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 accurate matrix multiplication. First we define a form which is expressed by an unevaluated summation of some floating-point numbers in order to have many bits. We call this form 'multiple floating-point numbers'. We propose an algorithm which computes matrix multiplication on this form and outputs an accurate result by mainly using Level 3 operations in BLAS. It is easy for the proposed algorithm to implement and compute in parallel. Numerical results are also presented in order to illustrate the efficiency of the proposed method.

1. Introduction

This paper is concerned with accurate matrix multiplication $A \cdot B$ for $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}$. Assume that each element of both input matrices has much significant bits than single or double precision floating-point numbers defined by IEEE 754. Our purpose is to obtain an accurate result of such matrix multiplication. Such matrix multiplication is required when we compute an inverse matrix of an ill-conditional coefficient matrix [4]. When numerical libraries supporting multi-precision floating-point arithmetic are used with this view, for example [11, 1], an accurate result can be obtained.

In our previous work [5], we defined 'the multiple floating-point numbers'. The number is represented by an unevaluated summation of usual floating-point numbers defined by IEEE 754. Therefore, it is possible for this format to have much precision. When each element in a matrix is represented by multiple floating-point numbers, we call the matrix a multiple floating-point matrix. We have developed an algorithm of computing dot products for such format by exploiting an accurate dot product algorithm [7]. This discussion can straightforwardly be extended to matrix multiplication so that we obtain an accurate result of a product of multiple floating-point matrices.

In this paper, we propose a different strategy to compute accurate matrix multiplication with multiple floating-point numbers by using mainly level 3 operation in optimized BLAS (Basic Linear Algebra Subprograms). For example, Goto BLAS, Intel Math Kernel Library and ATLAS are well-known as the optimized BLAS. It is particularly notable in such optimized BLAS that performance of 'gemm', routines for matrix multiplication, is nearly peak. Moreover, these routines are automatically parallelized in multi-threads environment. Dominant computations in the proposed method depend on such routines so that it receives much benefit from the routines in terms of the performance and the parallelization. At the end of this paper, numerical examples are shown to illustrate the efficiency of the proposed method.

2. Notation and multiple floating point numbers

In this section, we introduce notation and the definition of multiple floating-point numbers. All computations are performed by floating-point arithmetic defined by IEEE 754. In this paper, we use the double precision numbers and their arithmetic. Let $\mathbb{F}$ be a set of floating-point numbers and $\mathbf{u} = 2^{-53}$ be unit roundoff. MATLAB notation is used to describe algorithms for readability.

A normalized double precision floating-point number defined by IEEE 754 has 53 significant bits. Let $a, b \in \mathbb{F}$ and suppose a and b are not overlapped each other. An unevaluated summation $a + b$ has minimally 106 bits. Generally, let d be an unevaluated summation of floating-point numbers:

$$d = \sum_{i=1}^{n} d^{(i)}, \quad d^{(i)} \in \mathbb{F}$$

If all pairs of $d^{(i)}$ and $d^{(j)}$ are not overlapped each other and the following inequalities hold:

$$\mathbf{u}^{j-i}|a^{(i)}| \geq |a^{(j)}|, \quad i < j \tag{1}$$

Then we call d 'multiple floating-point numbers' [1]. From this definition, $d^{(1)}$ has leading 53 bits. Totally, d has minimally $53n$ bits.

[1] In some papers, it is also called non-overlapping expansion. Remark that there are some definitions for non-overlapping expansion.

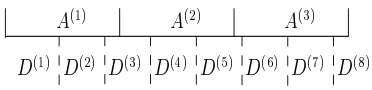

Figure 1: Relation of A and D.

Remark 1 *Even if we use multiple floating-point numbers, the maximum or minimum value is (almost) the same to that of a floating-point number since ranges of overflow and underflow of multiple floating-point number are the same to usual floating-point number.*

3. Proposed method

In this section, we propose the method which computes matrix multiplication for multiple-floating-point matrices by using level 3 operation in BLAS. Let A and B be multiple floating-point matrices:

$$A = \sum_{i=1}^{k} A^{(i)}, \quad B = \sum_{j=1}^{k} B^{(j)}, \quad A \in \mathbb{F}^{m\times n}, \quad B \in \mathbb{F}^{n\times p}$$

First, we transform A and B to a summation of floating-point matrices respectively such that

$$D = \sum_{i=1}^{r} D^{(i)} = \sum_{i=1}^{k} A^{(i)}, \quad E = \sum_{i=1}^{s} E^{(i)} = \sum_{j=1}^{k} B^{(i)}, \qquad (2)$$

where $r, s \geq k$ (see Figure 1). If we specialize our previous work [6], then we can construct splitting algorithms in order to satisfy (2) and

$$fl(D^{(i)}E^{(j)}) = D^{(i)}E^{(j)}, \quad 1 \leq i \leq r, \quad 1 \leq j \leq s. \qquad (3)$$

Then, we can compute matrix multiplication $A \cdot B$ as follows:

$$AB = \sum_{i=1}^{r} D^{(i)} \sum_{j=1}^{s} E^{(j)} \qquad (4)$$

After expanding (4), it involves matrix multiplication rs times. Therefore, matrix multiplication can be transformed into a unevaluated summation of rs floating-point matrices without rounding errors. If we use accurate summation algorithms [7, 8, 2] after this transformation, we can obtain an arbitrary accurate result.

We denote the splitting algorithm for this purpose as follows:

Algorithm 1 *Let A be a multiple floating-point matrix as*

$$A = \sum_{i=1}^{r} A^{(i)}, \quad A^{(i)} \in \mathbb{F}^{m\times n}.$$

The following algorithm transforms $\sum A^{(i)}$ to $\sum D^{(i)}$ in order to satisfy (3). In MATLAB notations, $A^{(i)} = A\{i\}$ (cell-array is used).

```
function D = Split_A(A)
  if  iscell(A) == 0,   A = {A};,   end
  p = size(A{1}, 2);
  Aindex = 1;
  n = length(A);
  count = 0;
  while norm(A{1}, inf) = 0
    μ = max(abs(A{1}), [], 2);
    t = 2.^(ceil(log2(μ)) + ceil((53 + log 2(p))/2));
    σ = repmat(t, 1, p);
    D{Aindex} = (A{1} + σ) − σ;
    A{1} = A{1} − D{Aindex};
    % data compression
    if mod(count, 2) == 1
      for i = 1 : n − 1
        [A{i}, A{i + 1}] = TwoSum(A{i}, A{i + 1});
      end
    end
    Aindex = Aindex + 1;
    count = count + 1;
  end
end
```

Here TwoSum is used in Algorithm 1. The following is the detail of this algorithm:

Algorithm 2 *For $a, b, x, y \in \mathbb{F}$, the following algorithm transforms $a + b$ into $x + y$ such that*

$$a + b = x + y, \quad x = fl(a + b), \quad \mathbf{u}|x| \geq |y|$$

where y holds the error of $fl(x + y)$ exactly.

```
function [x, y] = TwoSum(a, b)
x = a + b;
bvirt = x − a;
avirt = x − bvirt;
bround = b − bvirt;
around = a − avirt;
y = around + bround;
```

Next we present an algorithm which transforms $\sum B^{(i)}$ to $\sum E^{(i)}$ in order to satisfy (3).

Algorithm 3 *Let B be a multiple floating-point matrix as*

$$B = \sum_{i=1}^{r} B^{(i)}, \quad B^{(i)} \in \mathbb{F}^{n\times p}.$$

The following algorithm transforms $\sum B^{(i)}$ to $\sum E^{(i)}$ in or-

der to satisfy (3).

```
function E = Split_B(B)
  if iscell(B) == 0,  B = {B};,  end
  p = size(B{1}, 1);
  Bindex = 1;
  n = length(B);
  count = 0;
  while norm(B{1}, inf) = 0
    μ = max(abs(B{1}));
    t = 2.^(ceil(log2(μ)) + ceil((53 + log2(p))/2));
    σ = repmat(t, p, 1);
    E{Bindex} = (B{1} + σ) − σ;
    B{1} = B{1} − E{Bindex};
    % data compression
    if mod(count, 2) == 1
      for i = 1 : n − 1
        [B{i}, B{i + 1}] = TwoSum(B{i + 1}, B{i});
      end
    end
    Bindex = Bindex + 1;
    count = count + 1;
  end
end
```

The following is the proposed method of computing accurate matrix multiplication $A \cdot B$ for multiple floating-point matrices A and B.

Algorithm 4 *Let A and B be multiple floating-point matrices as*

$$A = \sum_{i=1}^{r} A^{(i)},\ B = \sum_{j=1}^{s} B^{(i)},\ A^{(i)} \in \mathbb{F}^{m\times n},\ B^{(i)} \in \mathbb{F}^{n\times p}.$$

The following algorithm computes matrix multiplication $A \cdot B$.

```
function C = mul_mul(A, B)
  D = Split_A(A);
  E = Split_B(B);
  Aindex = length(D);
  Bindex = length(E);
  index = 0;
  for i = 1 : Aindex
    for j = 1 : Bindex
      index = index + 1;
      G{index} = D{i} * E{j};
    end
  end
  C = Accrate_sum(G);
end
```

In Algorithm 4, Accrate_sum(G) computes the summation $\sum_{i=1}^{n} G\{i\}$ by algorithms in [9, 2, 7, 3].

The computations of matrix multiplication are dominant in our method in terms of computational costs. We just exploit the routine in optimized BLAS for such expensive computations. As another advantage, when we compute the matrix multiplication on multi-cores environment, the code for our algorithm needs not to be changed for parallelization. We only change the constant for number of threads so that it is easy for our method to perform parallel computations on the computational environment of the symmetrical multi-processor.

Remark 2 *If we implement accurate summation algorithms on MATLAB, the performance is very low due to the interpretation overhead. By using external interface, this point can be overcome.*

4. Numerical examples

In this section, we present numerical examples to illustrate the efficiency of the proposed method. Let A and B be represented by multiple floating-point matrices as

$$A = \sum_{i=1}^{k} A^{(i)}, \quad B = \sum_{j=1}^{k} B^{(j)}, \quad A^{(i)} \in \mathbb{F}^{n\times n}, \quad B^{(j)} \in \mathbb{F}^{n\times n}. \tag{5}$$

We check the performance of the following methods:

- MPFR [11] [2]
- the proposed method (Algorithm 4)

Numerical examples are tested on Ubuntu 7.10 (64 bit OS) with Xeon 2.5 GHz and MATLAB 2007b [3]. Amount of memory installed in the computer is 8 GByte. First, dimension of matrices n and a number of the summation k in (5) is set as $n = 500, 1000, 2000$ and $k = 2, 3, \ldots, 10$, respectively. We use `SumK` (described in [3]) as `Accrate_sum` in Algorithm 4. We set a precision as $53k$ in MPFR. Tables 1 to 3 shows the computing times for $n = 500, 1000, 2000$, respectively with various k. The computing times are measured in seconds. Remark that when we implement the examples by MPFR, we should first transform multiple floating-point matrices into the form of MPFR (mpfr_t). However, the cost of this transformation is almost negligible. The notation '$-$' means that the algorithm stops due to out of memory.

It can be confirmed from these tests that the proposed method works faster. As a drawback, the proposed method requires much amount of memory. When $n = 2000$ and $k \geq 6$, our method cannot work due to insufficient memory. If we set $k \geq 11$, $A^{(1)} =$ randn(n) and $B^{(1)} =$ randn(n), where randn(n) returns an n-by-n matrix containing pseudo-random values drawn from the standard normal distribution, then under flow may occur in Algorithm 4. In that case, our method may fail to output accurate result and the performance is significantly low due to treating unnormalized numbers. The routine in MPFR can work faster in such a condition.

[2] There is triple loops in a program of matrix multiplication. When we make the program for matrix multiplication on MPFR, we choose the suitable order of loop in terms of computing time.

[3] We use a single thread. We implement the code for MPFR to use MATLAB's external interface (mex) and compile it by GCC 4.1.3.

Table 1: Elapsed time (sec) for each method (n = 500)

k	mpfr	the proposed method	ratio
2	15.5	2.11	7.34
3	20.4	3.79	5.38
4	24.1	7.07	3.40
5	28.8	9.94	2.89
6	30.3	13.3	2.27
7	33.9	17.1	1.98
8	39.8	22.4	1.77
9	48.9	28.6	1.71
10	51.2	34.1	1.50

Table 2: Elapsed time (sec) for each method (n = 1000)

k	mpfr	the proposed method	ratio
2	121	15.2	7.9
3	160	29.4	5.44
4	187	44.2	4.23
5	222	62.3	3.56
6	234	94.0	2.48
7	264	121	2.18
8	307	156	1.96
9	353	193	1.82
10	402	229	1.75

5. Conclusion

We proposed the algorithm which computes accurate matrix multiplication. In our algorithm, there is a drawback for amount of required working memory. Our algorithm can work faster when the range of input data is suited for multiple-floating point numbers. Moreover, it is easy to implement for parallel computing. Even if an interpreted language like MATLAB is used, our method can work fast without external interface. Therefore, our algorithm works portably.

Acknowledgments

This research was partially supported by CREST program, Japan Science and Technology Agency (JST).

References

[1] David H. Bailey, "A Fortran-90 Based Multiprecision System", ACM Transactions on Mathematical Software, vol. 21, pp. 379-387, 1995.

[2] J. Demmel, Y. Hida, "Accurate and Efficient Floating Point Summation", SIAM J. Sci. Comput., vol. 25, pp. 1214–1248, 2003.

Table 3: Elapsed time (sec) for each method (n = 2000)

k	mpfr	the proposed method	ratio
2	963	98.7	9.75
3	1268	199	6.37
4	1471	301	4.88
5	1751	524	3.34
6	1847	-	-
7	2075	-	-
8	2430	-	-
9	2792	-	-
10	3209	-	-

[3] T. Ogita, S. M. Rump, S. Oishi, "Accurate sum and dot product", SIAM J. Sci. Comput., vol. 26, pp. 1955–1988, 2005.

[4] S. Oishi, K. Tanabe, T. Ogita, and S.M. Rump, "Convergence of Rump's method for inverting arbitrarily ill-conditioned matrices", J. Comput. Appl. Math., vol. 205, pp. 533-544, 2007.

[5] K. Ozaki, T. Ogita, S. M. Rump, S. Oishi, "Accurate Matrix Multiplication with Multiple Floating-point Numbers", Proceeding of 2007 International Symposium on Nonlinear Theory and its Applications, pp. 337-340, 2007.

[6] K. Ozaki, T. Ogita, S. M. Rump, S. Oishi, "Accurate Matrix Multiplication by using Level 3 BLAS Operation", Proceeding of 2008 International Symposium on Nonlinear Theory and its Applications, pp. 508-511, 2008.

[7] S. M. Rump, T. Ogita, and S. Oishi, "Accurate Floating-point Summation Part I: Faithful Rounding", SIAM J. Sci. Comput., vol. 31, pp. 189-224, 2008.

[8] S. M. Rump, T. Ogita, and S. Oishi, "Accurate Floating-point Summation Part II: Sign, K-fold Faithful and Rounding to Nearest", SIAM J. Sci. Comput., vol. 31, pp. 1269-1302, 2008.

[9] J. R. Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates", Discrete & Computational Geometry, vol. 18, pp. 305–363, 1997.

[10] MATLAB Programming version 7, the mathworks.

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