Journal of Computational and Applied Mathematics 288 (2015) 127–140

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Improvement of error-free splitting for accurate matrix multiplication

Katsuhisa Ozaki [a,d,*], Takeshi Ogita [b,d], Shin'ichi Oishi [c,d]

[a] Department of Mathematical Sciences, College of Systems Engineering and Science, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-shi 307-8570, Japan

[b] Division of Mathematical Sciences, School of Arts and Sciences, Tokyo Woman's Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan

[c] Faculty of Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan

[d] JST, CREST, Japan

ARTICLE INFO

Article history:
Received 12 October 2013

Keywords:
Matrix multiplication
Accurate computations
Interval arithmetic

ABSTRACT

Recently, new algorithms for accurate matrix multiplication have been developed by the authors. A characteristic of the algorithms is a high dependency on level-3 BLAS routines, which are highly optimized for several architectures. An error-free splitting for floating-point matrices is a key technique in the algorithms. In this paper, an improvement of the error-free splitting is focused on. It is shown by numerical examples that the accuracy of computed results of matrix products can be improved by the modified error-free splitting, compared to that by the previous algorithms.

1. Introduction

This paper is concerned with accurate matrix multiplication. Floating-point arithmetic as defined by IEEE 754 [1,2] is performed very fast on recent architectures. On the other hand, since the significand of the floating-point numbers is finite, floating-point arithmetic may cause rounding errors on each arithmetic operation. If rounding errors accumulate, then inaccurate results may be obtained. To overcome this problem on matrix multiplication, there are several possibilities:

- multiple precision libraries [3–6]
- mixed precision libraries [7,8]
- accurate dot product or summation algorithms [9–13].[1]

Recently, we have developed accurate algorithms for floating-point matrix multiplication [14,15]. Our algorithms involve several floating-point matrix products which can be computed by functions supported in BLAS (Basic Linear Algebra Subprograms). If so-called optimized BLAS is used, for example, OpenBLAS [16] based on GotoBLAS2 [17], Intel Math Kernel Library and so forth, then the algorithms can receive the benefit of performance from the BLAS since the performance of functions for matrix multiplication in such BLAS is nearly peak. Since the level 3 fraction [18], the amount of matrix multiplication in a given algorithm, is very high, our algorithms produce accurate results very fast.

* Corresponding author at: Department of Mathematical Sciences, College of Systems Engineering and Science, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-shi 307-8570, Japan.

E-mail address: ozaki@sic.shibaura-it.ac.jp (K. Ozaki).

[1] Since a dot product can be transformed into an unevaluated sum of floating-point numbers by so-called error-free transformation, accurate summation algorithms can be applied into the dot product. See [9].

http://dx.doi.org/10.1016/j.cam.2015.04.010
0377-0427/

A key technique of the algorithms is an error-free splitting for floating-point matrices which transforms a floating-point matrix into an unevaluated sum of several floating-point matrices. We focus on the case where given two floating-point matrices are divided into unevaluated sums of two floating-point matrices, respectively. Let $\mathbb{F}$ be the set of floating-point numbers as defined by the IEEE 754 standard [1,2]. For $A \in \mathbb{F}^{m\times n}$ and $B \in \mathbb{F}^{n\times p}$, we split A and B such that $A = A^{(1)}+A^{(2)}$, $B = B^{(1)} + B^{(2)}$, where $|a_{ij}^{(1)}| \geq |a_{ij}^{(2)}|$ and $|b_{ij}^{(1)}| \geq |b_{ij}^{(2)}|$ except $a_{ij}^{(1)} = 0$ and $b_{ij}^{(1)} = 0$. Moreover, even if we evaluate $A^{(1)}B^{(1)}$ by the floating-point arithmetic, no rounding error occurs. Therefore, it is shown in [14,15] that the accuracy of the computed result by $A^{(1)}B^{(1)} + A^{(1)}B^{(2)} + A^{(2)}B$ is better than that by the pure floating-point evaluation of AB in many cases.

In this paper, we improve an algorithm which produces $A^{(1)}$, $A^{(2)}$, $B^{(1)}$ and $B^{(2)}$ from A and B. As a result, the accuracy of the approximation of AB by the new algorithm is improved, compared to the previous algorithms in many cases. Finally, numerical results are presented to illustrate the efficiency of the proposed algorithm.

2. Notation and previous work

In this section, notation used in this paper is first introduced. The notation $fl(\cdots)$ means that each operation in the parenthesis is performed by floating-point arithmetic defined by the IEEE 754-2008 standard as follows: $fl(\cdots)$: roundTiesToEven, $fl_{\triangle}(\cdots)$: roundTowardPositive and $fl_{\triangledown}(\cdots)$: roundTowardNegative, respectively. Let $\mathbf{u}$ be the relative rounding error unit, especially, $\mathbf{u} = 2^{-53}$ for binary64 (so-called double precision). Assume that neither overflow nor underflow occurs in $fl(\cdots)$. Let $\mathbf{e}$ denote the vector of all one's with appropriate length. For $a \in \mathbb{R}$, $\mathtt{ufp}(a)$ denotes the unit in the first place [11], namely, the leading bit of the binary representation of a. For $0 \neq b \in \mathbb{F}$,

$$\mathtt{ufp}(b) = 2^{\lfloor \log_2 |b| \rfloor}, \quad b \in 2\mathbf{u}\cdot \mathtt{ufp}(b)\mathbb{Z}. \tag{1}$$

Exceptionally, we define $\mathtt{ufp}(0) := 0$. Note that $\mathtt{ufp}(b)$ is easily calculated by three floating-point operations without evaluating the logarithm [19]. MATLAB-like notation [20] is used for a representation of algorithms for readability. For $x, y \in \mathbb{F}^n$, $x > y$ means that componentwise inequalities are always satisfied, namely, $x_i > y_i$ for $1 \leq i \leq n$. $|x|$ means that $|x| = (|x_1|, |x_2|, \ldots, |x_n|)^T$. Similar notation is used for matrices.

We briefly introduce our previous algorithm: Algorithm 4.1 in [15], $\mathtt{Accmul}(A, B, 2, \delta)$. For $A \in \mathbb{F}^{m\times n}$ and $B \in \mathbb{F}^{n\times p}$, the purpose is to obtain $C \in \mathbb{F}^{m\times p}$ as an accurate approximation of a matrix product AB with reasonable cost. Let a constant β be

$$\beta = \left\lceil \frac{\log_2 n - \log_2 \mathbf{u}}{2} \right\rceil.$$

Two vectors $\sigma \in \mathbb{F}^m$ and $\tau \in \mathbb{F}^p$ are defined by

$$\sigma_i = 2^{\beta}\cdot 2^{v_i}, \qquad \tau_j = 2^{\beta}\cdot 2^{w_j}, \tag{2}$$

where two vectors $v \in \mathbb{F}^m$ and $w \in \mathbb{F}^p$ are defined by

$$v_i = \lceil \log_2 \max_{1\leq j\leq n} |a_{ij}| \rceil \quad \text{for} \quad \max_{1\leq j\leq n} |a_{ij}| \neq 0, \qquad v_i = 0 \quad \text{for} \quad \max_{1\leq j\leq n} |a_{ij}| = 0,$$

$$w_j = \lceil \log_2 \max_{1\leq i\leq n} |b_{ij}| \rceil \quad \text{for} \quad \max_{1\leq i\leq n} |b_{ij}| \neq 0, \qquad w_j = 0 \quad \text{for} \quad \max_{1\leq i\leq n} |b_{ij}| = 0.$$

The following algorithm implemented in INTLAB [21] is useful for the calculation of 2^{v_i} and 2^{w_j} in (2).

Algorithm 1 (*Rump*)**.** For $g \in \mathbb{F}^n$, the following algorithm produces a vector h such that $h_i = 2^{\lceil \log_2 |g_i| \rceil}$ ($g_i \neq 0$) and $h_i = 0$ ($g_i = 0$).

```
function h = NextPowerTwo(g)
  q = fl(u^{-1} * g);
  h = fl(abs((q - g) - q));
end
```

We obtain $A^{(1)}$ and $A^{(2)}$ by

$$A^{(1)} = fl\left((A + \sigma\cdot \mathbf{e}^T) - \sigma\cdot \mathbf{e}^T\right), \qquad A^{(2)} = fl\left(A - A^{(1)}\right).$$

Similarly, B is divided into $B^{(1)}$ and $B^{(2)}$ by

$$B^{(1)} = fl\left((B + \mathbf{e}\cdot \tau^T) - \mathbf{e}\cdot \tau^T\right), \qquad B^{(2)} = fl\left(B - B^{(1)}\right).$$

Then, it is proved in [15] that

$$A = A^{(1)} + A^{(2)}, \qquad B = B^{(1)} + B^{(2)}, \qquad A^{(1)}B^{(1)} = fl(A^{(1)}B^{(1)}). \tag{3}$$

Therefore,

$$AB = (A^{(1)} + A^{(2)})(B^{(1)} + B^{(2)}) = A^{(1)}B^{(1)} + A^{(1)}B^{(2)} + A^{(2)}B.$$

We obtain an approximate result of the matrix multiplication AB by

$$AB \approx fl(A^{(1)}B^{(1)} + (A^{(1)}B^{(2)} + A^{(2)}B)). \tag{4}$$

Finally, we write the algorithm in [15] which computes (4):