Interval Matrix Multiplication Using Fast Low-precision Arithmetic on GPU

Katsuhisa Ozaki[1], Takeshi Ogita[2], Daichi Mukunoki[3]

[1]*Department of Mathematical Sciences, Shibaura Institute of Technology, ozaki@sic.shibaura-it.ac.jp*

[2]*Division of Mathematical Sciences, Tokyo Woman's Christian University, ogita@lab.twcu.ac.jp*

[3]*RIKEN Center for Computational Science, daichi.mukunoki@riken.jp*

Abstract. This paper discusses interval arithmetic, in particular, interval matrix multiplication using Graphics Processing Units (GPUs). Interval matrix multiplication plays an important role in verification methods for numerical linear algebra. For some GPUs, single-precision arithmetic is performed much faster than double-precision arithmetic. In this paper, we propose fast methods for performing interval matrix multiplication based on an error-free transformation of matrix multiplication that uses fast single-precision arithmetic and Tensor Cores on a GPU. Numerical examples also illustrate the efficiency of the proposed methods in terms of tightness of intervals and computational speed.

Keywords: interval arithmetic, matrix multiplication, GPU computing, mixed-precision computations

1. Introduction

The concept of an interval and its arithmetic, e.g., (Sunaga, 1958; Moore, 1966) has been widely applied to scientific problems. In this paper, we focused on interval matrix multiplication using a Graphics Processing Unit (GPU). Interval matrix multiplication plays crucial role in verification methods for numerical linear algebra, for example, for matrix determinants (Rump, 2005), matrix square roots (Frommer and Hashemi, 2010), eigenproblems (Oishi, 2001; Miyajima et al., 2010; Miyajima, 2012; Hoshi et al., 2020) and Sylvester matrix equations (Frommer and Hashemi, 2012; Miyajima, 2013). Several fast methods (Rump, 1999; Ogita and Oishi, 2005; Rump, 2012; Ozaki et al., 2012; Ozaki et al., 2015) for performing interval matrix multiplication exist, though the tightness of computed intervals and computing times continue to pose problems.

In this study, we propose fast computation methods for utilizing single-precision arithmetic (binary32) in IEEE 754 (IEEE, 2008) in interval matrix multiplication that is originally computed with double-precision arithmetic (binary64) in IEEE 754. This method is based on the error-free transformations published by (Ozaki et al., 2012), and instead of binary32 arithmetic, it also allows one to use Tensor Cores, which is a binary16/binary32 mixed-precision matrix engine available on recent NVIDIA GPUs. In high performance computing, there has been research on utilizing low

precision arithmetic units, which are available in the market at low cost due to the huge demand, for scientific and engineering computations. Examples include the use of binary32 units originally developed for image and media processing (e.g., Cell/B.E. and GPU with CUDA, both appeared in 2006), and the use of fast low-precision arithmetic units for artificial intelligence (e.g., Tensor Cores, Google TPUs), which has been developing remarkably in recent years. Following this trend, this study aims to make effective use of low-cost computing resources by utilizing low-precision arithmetic, and to expand the potential of low-precision arithmetic itself.

2. Error-Free Transformation Using Low-Precision Arithmetic

Let $\mathbb{F}_q$ ($\subset \mathbb{R}$) be a set of binary floating-point numbers with q bit in IEEE 754, for example, $\mathbb{F}_{64}$ is a set of binary64 floating-point numbers. Notation $\mathrm{fl}_q(\cdot)$ indicates that all operations in the parenthesis are evaluated using q bit floating-point arithmetic. The rounding mode of $\mathrm{fl}_q(\cdot)$ is rounding to nearest mode. For simplicity, we assumed that neither overflow nor underflow occurs in $\mathrm{fl}_q(\cdot)$. Let $\mathbf{u}_q$ be the roundoff unit for q bit floating-point numbers, i.e., $\mathbf{u}_{32} = 2^{-24}$ and $\mathbf{u}_{64} = 2^{-53}$ in IEEE 754. For example for $\mathbb{F}_{64}$, the maximum floating-point number is $(1 - \mathbf{u}_{64})2^{1024}$ and the minimum number in normalized positive floating-point numbers is 2^{-1022}. Hence, overflow or underflow is rare to occur. Once overflow occurs for a computations of dot product, a result is one of `Inf`, $-$`Inf`, or `NaN`. Therefore, occurrence of overflow is easy to find from a computed result. It is also possible to treat the case of underflow by extending rounding error analysis. A function $\mathtt{RN}_r(a)$ for $a \in \mathbb{R}$ produces a nearest floating-point number in $\mathbb{F}_r$ to a real number a. Note that for $a \in \mathbb{R}$

$$\mathtt{RN}_r(a) = a + \delta, \quad |\delta| \le \mathbf{u}_r|a| \tag{1}$$

holds true except in the case that $|a|$ is smaller than the smallest positive number in the normalized floating-point number. For $x \in \mathbb{R}^n$, $|x|$ indicates $(|x_1|, |x_2|, \ldots, |x_n|)^T$. The manner $|\cdot|$ is straightforwardly extended to a matrix. A function $\mathtt{ufp}(a)$ for $a \in \mathbb{R}$, the unit in the first place of the binary representation of a, is defined as

$$\mathtt{ufp}(a) := \begin{cases} 0 & a = 0 \\ 2^{\lfloor \log_2 |a| \rfloor} & \text{otherwise} \end{cases}.$$

We briefly review an error-free transformation of matrix multiplication (Ozaki et al., 2012). Applying Algorithm 3 in (Ozaki et al., 2012) for two given matrices $A \in \mathbb{F}_q^{m\times n}$ and $B \in \mathbb{F}_q^{n\times p}$ produces $A^{(s)}$ and $B^{(t)}$, such that

$$A = \sum_{s=1}^{n_A} A^{(s)}, \quad n_A \in \mathbb{N}, \quad A^{(s)} \in \mathbb{F}_q^{m\times n}, \ 1 \le s \le n_A \tag{2}$$

$$B = \sum_{t=1}^{n_B} B^{(t)}, \quad n_B \in \mathbb{N}, \quad B^{(t)} \in \mathbb{F}_q^{n\times p}, \ 1 \le t \le n_B \tag{3}$$

in order to satisfy

$$A^{(i)}B^{(j)} = \mathrm{fl}_q\left(A^{(i)}B^{(j)}\right), \quad 1 \le i \le n_A, \quad 1 \le j \le n_B.$$

We assume that a divide and conquer method, such as Strassen's method (Strassen, 1969) or Winograd's method (Winograd, 1971), were not applied to matrix multiplication. The constants n_A and n_B depend on the inner dimension n and the difference in the magnitude of elements in the rows of A and columns of B. Since no rounding error exists for the floating-point evaluation of $A^{(i)}B^{(j)}$ for all (i,j) pairs, AB can be transformed into an unevaluated sum of n_An_B floating-point matrices after computing $\mathrm{fl}_q\left(A^{(i)}B^{(j)}\right)$ for all (i,j) pairs.

In this paper, we propose an error-free transformation of AB using r bit ($r < q$) floating-point arithmetic. A typical example is $r = 32$ (binary32) and $q = 64$ (binary64) in IEEE 754. We assume that when $A \in \mathbb{F}_q^{m\times n}$ and $B \in \mathbb{F}_q^{n\times p}$ are rounded to $A' \in \mathbb{F}_r^{m\times n}$ and $B' \in \mathbb{F}_r^{n\times p}$, respectively, neither overflow nor underflow occurs in the rounding. Let $\underline{A}^{(1)} := A$ and $\underline{B}^{(1)} := B$. We define a constant β as

$$\beta := \left\lceil -\log_2 \mathbf{u}_q - \frac{-\log_2 \mathbf{u}_r - \log_2 n}{2} \right\rceil. \tag{4}$$

We set two vectors $v^{(k)} \in \mathbb{F}_q^m$ and $w^{(k)} \in \mathbb{F}_q^p$ for $k = 1$. If $\max\limits_{1\le j\le n} \left|\underline{a}_{ij}^{(k)}\right| = 0$, then we define $v_i^{(k)} := 0$. Similarly, $w_j^{(k)} := 0$ in the case of $\max\limits_{1\le i\le n} \left|\underline{b}_{ij}^{(k)}\right| = 0$. Otherwise, we set

$$v_i^{(k)} := \left\lceil \log_2 \max_{1\le j\le n} \left|\underline{a}_{ij}^{(k)}\right| \right\rceil, \quad w_j^{(k)} := \left\lceil \log_2 \max_{1\le i\le n} \left|\underline{b}_{ij}^{(k)}\right| \right\rceil. \tag{5}$$

Next, vectors $\sigma^{(k)} \in \mathbb{F}_q^m$ and $\tau^{(k)} \in \mathbb{F}_q^p$ are defined as

$$\sigma_i^{(k)} := \mathrm{fl}_q\left(0.75 \cdot 2^{\beta} \cdot 2^{v_i^{(k)}}\right), \quad \tau_j^{(k)} := \mathrm{fl}_q\left(0.75 \cdot 2^{\beta} \cdot 2^{w_j^{(k)}}\right) \tag{6}$$

using (4) and (5). Then, $A^{(k)}$, $\underline{A}^{(k+1)}$, $B^{(k)}$, $\underline{B}^{(k+1)}$ are obtained by

$$\begin{aligned} a_{ij}^{(k)} &:= \mathrm{fl}_q\left(\left(\sigma_i^{(k)} + \underline{a}_{ij}^{(k)}\right) - \sigma_i^{(k)}\right), \quad \underline{a}_{ij}^{(k+1)} := \mathrm{fl}_q\left(\underline{a}_{ij}^{(k)} - a_{ij}^{(k)}\right), \\ b_{ij}^{(k)} &:= \mathrm{fl}_q\left(\left(\tau_j^{(k)} + \underline{b}_{ij}^{(k)}\right) - \tau_j^{(k)}\right), \quad \underline{b}_{ij}^{(k+1)} := \mathrm{fl}_q\left(\underline{b}_{ij}^{(k)} - b_{ij}^{(k)}\right). \end{aligned} \tag{7}$$

If we compute (5), (6) and (7) from $k = 1, 2, \ldots$, then there exist $n'_A, n'_B \in \mathbb{N}$ such that

$$A = \sum_{s=1}^{n'_A} A^{(s)}, \quad B = \sum_{t=1}^{n'_B} B^{(t)}, \quad \underline{A}^{(n'_A+1)} = \mathbf{O}_{mn}, \quad \underline{B}^{(n'_B+1)} = \mathbf{O}_{np}, \tag{8}$$

where $\mathbf{O}_{mn}$ is the m-by-n zero matrix. For the matrices in (8),

$$A^{(i)} \in \mathbb{F}_r^{m\times n}, \quad B^{(j)} \in \mathbb{F}_r^{n\times p}, \quad A^{(i)}B^{(j)} = \mathrm{fl}_r\left(A^{(i)}B^{(j)}\right), \quad 1 \le i \le n'_A, \quad 1 \le j \le n'_B \tag{9}$$

are satisfied. This means that

- $\mathtt{RN}_r\left(A^{(i)}\right) = A^{(i)}$ and $\mathtt{RN}_r\left(B^{(j)}\right) = B^{(j)}$, and

– if we use r bit floating-point arithmetic for $A^{(i)}B^{(j)}$, a rounding error never occurs in the evaluation.

These are proved in the subsequent section. Hence, AB is transformed into an unevaluated $n'_A n'_B$ sums of floating-point matrices such that

$$AB = \sum_{r=1}^{n'_A} \sum_{s=1}^{n'_B} A^{(r)}B^{(s)} = \sum_{r=1}^{n'_A} \sum_{s=1}^{n'_B} \mathrm{fl}_r \left(A^{(r)}B^{(s)} \right). \tag{10}$$

If we obtain an accurate approximation of AB using the error-free transformation, we perform three steps as follows:

Step 1 Split matrices: $A = A^{(1)} + \ldots + A^{(n'_A)}$ and $B = B^{(1)} + \ldots + B^{(n'_B)}$ using (5), (6) and (7).

Step 2 Compute matrix multiplications $A^{(i)}B^{(j)},\ 1 \le i \le n'_A,\ 1 \le j \le n'_B$.

Step 3 Compute the summation in (10).

For Step 3, a user can use accurate summation algorithms, e.g., (Rump et al., 2008; Ogita et al., 2005; Yamanaka et al., 2008; Rump, 2009; Demmel and Hida, 2004; Zhu and Hayes, 2009; Zhu and Hayes, 2010; Rump et al., 2008). Or, pure floating-point arithmetic with proper order of summation can produce an accurate numerical result in many cases (Kazal et al.,). In these steps, the cost of Step 2 is the most expensive in many cases. Compared with (2) and (3), $n'_A n'_B$ is larger than $n_A n_B$. However, if r bit arithmetic is performed very quickly compared with q bit arithmetic, the computation of error-free transformation using low-precision arithmetic is finished quickly. In fact, we observed that matrix multiplication with binary32 worked more than 15 times faster than that with binary64 on the Quadro RTX 5000 by NVIDIA. This benchmark will be shown in Section 5.

3. Proof for the Error-Free Transformation

In this section, the proof of (9) is derived. We first introduce a basic lemma for floating-point arithmetic, which is derived from the definition of the floating-point number defined by IEEE 754.

Lemma 1. *For $a \in \mathbb{R}$, assume that $|a|$ is a multiple of the minimum positive floating-point number, and less than the maximum floating-point number. If we find $\mathbb{N} \cup \{0\} \ni k \le \mathbf{u}_q^{-1}$ and s (a power of two) such that $s\mathbb{Z} \ni |a| \le ks$, then $a \in \mathbb{F}_q$.*

The next lemma was derived by Minamihata et al. (Minamihata et al., 2016).

Lemma 2. *For $x \in \mathbb{F}_q$ and $M \in \mathbb{N}$, let*

$$\sigma := \begin{cases} 0.75 \cdot 2^M \cdot 2^{\lceil \log_2 |x| \rceil} & x \neq 0 \\ 0 & x = 0 \end{cases}.$$

If x_1 and x_2 are obtained as follows:

$$x_1 = \mathrm{fl}_q \left((\sigma + x) - \sigma \right), \quad x_2 = \mathrm{fl}_q \left(x - x_1 \right),$$

then

$$x = x_1 + x_2, \quad |x_1| \le \mathtt{ufp}(\sigma)2^{-M}, \quad x_1 \in \mathbf{u}_q\mathtt{ufp}(\sigma)\mathbb{Z}$$

are satisfied.

Applying Lemma 2 into the computations (4), (5), (6) and (7) yields

$$\left|a_{ij}^{(k)}\right| \le 2^{-\beta}\mathtt{ufp}\left(\sigma_i^{(k)}\right), \quad \left|b_{ij}^{(\ell)}\right| \le 2^{-\beta}\mathtt{ufp}\left(\tau_j^{(\ell)}\right), \quad a_{ij}^{(k)} \in \mathbf{u}_q\mathtt{ufp}\left(\sigma_i^{(k)}\right)\mathbb{Z}, \quad b_{ij}^{(\ell)} \in \mathbf{u}_q\mathtt{ufp}\left(\tau_j^{(\ell)}\right)\mathbb{Z}. \tag{11}$$

We use these properties for the following theorems.

Theorem 1. *For the matrices $A^{(i)}$ and $B^{(j)}$ in (9), that are generated by (4), (5), (6) and (7),*

$$A^{(i)} \in \mathbb{F}_r^{m\times n}, \quad B^{(j)} \in \mathbb{F}_r^{n\times p} \tag{12}$$

are satisfied for $1 \le i \le n'_A$ and $1 \le j \le n'_B$.

Proof. From the definition of β in (4), we have $2^{-\beta} < \mathbf{u}_r^{-1}\mathbf{u}_q$. This and (11) yield

$$\begin{aligned}
\mathbf{u}_q \cdot \mathtt{ufp}\left(\sigma_i^{(k)}\right)\mathbb{Z} \ \ni\ |a_{ij}^{(k)}| \le 2^{-\beta}\mathtt{ufp}\left(\sigma_i^{(k)}\right) < \mathbf{u}_r^{-1}\mathbf{u}_q\mathtt{ufp}\left(\sigma_i^{(k)}\right),\\
\mathbf{u}_q \cdot \mathtt{ufp}\left(\tau_j^{(k)}\right)\mathbb{Z} \ \ni\ |b_{ij}^{(k)}| \le 2^{-\beta}\mathtt{ufp}\left(\tau_j^{(k)}\right) < \mathbf{u}_r^{-1}\mathbf{u}_q\mathtt{ufp}\left(\tau_j^{(k)}\right).
\end{aligned}$$

Therefore, (12) is satisfied from Lemma 1. □

Theorem 2. *For the matrices $A^{(k)}$ and $B^{(\ell)}$ in (9),*

$$A^{(k)}B^{(\ell)} = \mathrm{fl}_r\left(A^{(k)}B^{(\ell)}\right) \tag{13}$$

are satisfied for $1 \le k \le n'_A,\ 1 \le \ell \le n'_B$.

Proof. From the definition of β in (4),

$$2^{-2\beta} = \frac{1}{2^{2\beta}} = \frac{1}{2^{2\left\lceil -\log_2 \mathbf{u}_q - \frac{-\log_2 \mathbf{u}_r - \log_2 n}{2}\right\rceil}} \le \frac{1}{2^{-2\log_2 \mathbf{u}_q + \log_2 \mathbf{u}_r + \log_2 n}} = \frac{1}{n}\mathbf{u}_r^{-1}\mathbf{u}_q^2 \tag{14}$$

holds true. Applying (11) yields

$$a_{iz}^{(k)}b_{zj}^{(\ell)} \in \mathbf{u}_q^2 \cdot \mathtt{ufp}\left(\sigma_i^{(k)}\right)\mathtt{ufp}\left(\tau_j^{(\ell)}\right)\mathbb{Z}$$

for all i, z, j, k, and ℓ. Using this, (11) and (14) gives

$$\begin{aligned}
\mathbf{u}_q^2\mathtt{ufp}\left(\sigma_i^{(k)}\right)\mathtt{ufp}\left(\tau_j^{(\ell)}\right)\mathbb{Z} \ &\ni\ \sum_{z=1}^{n} a_{iz}^{(k)}b_{zj}^{(\ell)} \le \sum_{z=1}^{n} |a_{iz}^{(k)}||b_{zj}^{(\ell)}|\\
&\le\ n2^{-2\beta}\mathtt{ufp}\left(\sigma_i^{(k)}\right)\mathtt{ufp}\left(\tau_j^{(\ell)}\right) \le \mathbf{u}_r^{-1}\mathbf{u}_q^2\mathtt{ufp}\left(\sigma_i^{(k)}\right)\mathtt{ufp}\left(\tau_j^{(\ell)}\right).
\end{aligned}$$

Therefore, (13) is proved from Lemma 1. □

A computed result of the dot product depends on a computational order, for example, recursive, block-wise, and pair-wise orders, due to rounding errors. Note that Theorem 2 is valid for such orders of computation. This can be proved using an induction on binary trees as in (Jeannerod and Rump, 2013).

4. Application to Interval Matrix Multiplication

4.1. Basics of Methods for Interval Matrix Multiplication

Let $\mathbb{IR}$ be a set of real intervals. We define an inf-sup form $[a, b]$ with real numbers $a, b \in \mathbb{R}$ $(a \leq b)$ and a mid-rad form $\langle c, r\rangle$ with real numbers $c, r \in \mathbb{R}$ $(r \geq 0)$:

$$[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}, \quad \langle c, r\rangle = \{x \in \mathbb{R} \mid c - r \leq x \leq c + r\}.$$

These intervals are straightforwardly extended to vectors and matrices. We introduced floating-point arithmetic with directed rounding. Let

- $\mathrm{fl}_{\triangledown,q}(\cdot)$: a computed result using rounding downward mode (roundTowardNegative) using q bit floating-point arithmetic
- $\mathrm{fl}_{\triangle,q}(\cdot)$: a computed result using rounding upward mode (roundTowardPositive) using q bit floating-point arithmetic

For $A_m, A_r \in \mathbb{F}_q^{m\times n}$ and $B_m, B_r \in \mathbb{F}_q^{n\times p}$ ($A_r = |A_r|$ and $B_r = |B_r|$), an interval matrix multiplication $\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle$ is examined. It is well known, e,g, (Rump, 1999), that

$$\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \subseteq \langle A_m B_m,\ T_1\rangle, \quad T_1 = |A_m|B_r + A_r(|B_m| + B_r). \tag{15}$$

Based on (15), Rump proposed the following method (Rump, 1999)

$$\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \subseteq [\mathrm{fl}_{\triangledown,q}(A_m B_m - T_1'),\ \mathrm{fl}_{\triangle,q}(A_m B_m + T_1')],$$

where

$$T_1' = \mathrm{fl}_{\triangle,q}(|A_m|B_r + A_r(|B_m| + B_r)).$$

In total, four matrix multiplications are necessary to compute the enclosure of interval matrix multiplication. Ozaki et al. (Ozaki et al., 2012) and Rump (Rump, 2012) have proposed a method with three matrix multiplications. For a computed result of matrix multiplication,

$$|A_m B_m - \mathrm{fl}_q(A_m B_m)| \leq n\mathbf{u}_q |A_m||B_m| \tag{16}$$

is satisfied barring overflow and underflow (Jeannerod and Rump, 2013). Then, we have

$$A_m B_m \in \langle \mathrm{fl}_q(A_m B_m),\ n\mathbf{u}_q |A||B|\rangle. \tag{17}$$

From (17) and (15),

$$\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \subset \langle \mathrm{fl}_q(A_m B_m),\ T_2\rangle \tag{18}$$

is satisfied, where

$$T_2 = |A_m|(n\mathbf{u}_q|B_m| + B_r) + A_r(|B_m| + B_r). \tag{19}$$

Using direct rounding, $\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle$ is enclosed by

$$\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \subset \langle \mathrm{fl}_q(A_m B_m),\ T_2'\rangle, \tag{20}$$

where

$$T_2' = \mathrm{fl}_{\triangle,q}(|A_m|(n\mathbf{u}_q|B_m| + B_r) + A_r(|B_m| + B_r)). \tag{21}$$

The paper (Ozaki et al., 2015) have shown that if low-precision arithmetic is used to obtain the upper bounds of T_1 and T_2, the overestimation of the radius is almost negligible in many cases. Let $A_m', A_r' \in \mathbb{F}_r^{m\times n}$ and $B_m', B_r' \in \mathbb{F}_r^{n\times p}$ such that

$$|A_m| \le A_m' \in \mathbb{F}_r^{m\times n}, \quad A_r \le A_r' \in \mathbb{F}_r^{m\times n}, \quad |B_m| \le B_m' \in \mathbb{F}_r^{n\times p}, \quad B_r \le B_r' \in \mathbb{F}_r^{n\times p}.$$

Because

$$\begin{aligned} T_1 &= |A_m|B_r + A_r(|B_m| + B_r) \le A_m'B_r' + A_r'(B_m' + B_r') \le \mathrm{fl}_{\triangle,r}(A_m'B_r' + A_r'(B_m' + B_r')) := T_1'', \\ T_2 &= |A_m|(n\mathbf{u}_q|B_m| + B_r) + A_r(|B_m| + B_r) \le \mathrm{fl}_{\triangle,r}(A_m'(n\mathbf{u}_q B_m' + B_r') + A_r'(B_m' + B_r')) := T_2'', \end{aligned}$$

the interval matrix multiplication $\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle$ is enclosed by

$$\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \ \subseteq\ [\mathrm{fl}_{\triangledown,q}(A_m B_m - T_1''),\ \mathrm{fl}_{\triangle,q}(A_m B_m + T_1'')], \tag{22}$$

$$\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \ \subseteq\ \langle \mathrm{fl}_q(A_m B_m),\ T_2''\rangle. \tag{23}$$

Obtaining the interval in (22) involves two matrix multiplications with q bit arithmetic and two matrix multiplications with r bit arithmetic. For (23), the cost is one matrix multiplication with q bit arithmetic and two matrix multiplications with r bit arithmetic.

Another possibility is to use Ogita and Oishi's technique. For $X = |X| \in \mathbb{F}_q^{m\times n}$ and $Y = |Y| \in \mathbb{F}_q^{n\times p}$, set four vectors $g \in \mathbb{R}^n$, $h \in \mathbb{R}^n$, $e \in \mathbb{F}^m$, and $f \in \mathbb{F}^p$ as follows

$$g_j = \max_i |X_{ij}|, \quad h_i = \max_j |Y_{ij}|, \quad e = (1, 1, \ldots, 1)^T \in \mathbb{F}^m, \quad f = (1, 1, \ldots, 1)^T \in \mathbb{F}^p. \tag{24}$$

Then, XY is bounded by

$$XY \le \min\left(e(gY), (Xh)f^T\right) \le \min\left(\mathrm{fl}_{\triangle,q}\left(e(gY)\right), \mathrm{fl}_{\triangle,q}\left((Xh)f^T\right)\right) =: F, \tag{25}$$

where $\min(C, D)$ returns a matrix with the smallest elements taken from $C \in \mathbb{F}^{n\times n}$ or $D \in \mathbb{F}^{n\times n}$. The result of a product of two positive matrices can be bounded without computing full matrix multiplication. We denoted a function $F = \mathtt{fn}(X, Y, q)$ for computation (25). Using this function, we have

$$|A_m|B_r + A_r(|B_m| + B_r) \le \mathrm{fl}_{\triangle,q}\left(\mathtt{fn}(|A_m|, B_r, q) + \mathtt{fn}(A_r, \mathrm{fl}_{\triangle,q}(|B_m| + B_r), q)\right) := T_1''',$$

and

$$\begin{aligned} & |A_m|(n\mathbf{u}_q|B_m| + B_r) + A_r(|B_m| + B_r) \\ \le\ & \mathtt{fn}(|A_m|, \mathrm{fl}_{\triangle,q}\left(n\mathbf{u}_q|B_m| + B_r\right), q) + \mathtt{fn}(A_r, \mathrm{fl}_{\triangle,q}\left(|B_m| + B_r\right), q) \\ \le\ & \mathrm{fl}_{\triangle,q}\left(\mathtt{fn}(|A_m|, \mathrm{fl}_{\triangle,q}\left(n\mathbf{u}_q|B_m| + B_r\right), q) + \mathtt{fn}(A_r, \mathrm{fl}_{\triangle,q}\left(|B_m| + B_r\right), q)\right) =: T_2'''. \end{aligned}$$

Then,

$$\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \ \subseteq\ [\mathrm{fl}_{\bigtriangledown,q}(A_mB_m - T_1'''),\ \mathrm{fl}_{\bigtriangleup,q}(A_mB_m + T_1''')], \tag{26}$$

$$\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \ \subseteq\ \langle \mathrm{fl}_q(A_mB_m),\ T_2'''\rangle. \tag{27}$$

Method (26) with two matrix multiplications was proposed by Ogita and Oishi (Ogita and Oishi, 2005), and method (27) with a matrix multiplication was proposed by Ozaki et al. (Ozaki et al., 2012).

4.2. Proposed Methods

We propose a method based on (15). Because we cannot switch rounding modes on GPUs for already compiled BLAS routines, for example, $\mathrm{fl}_{\bigtriangledown,q}(A_mB_m - T_1''')$, $\mathrm{fl}_{\bigtriangleup,q}(A_mB_m + T_1''')$ in (26) could not be computed using functions in cuBLAS (NVIDIA, 2007). Therefore, we obtained the enclosure of interval matrix multiplication by only rounding to nearest mode. For $0 \le a, b \in \mathbb{F}_q$,

$$a + b \le \mathrm{fl}_q((\theta_q(a + b)), \quad ab \le \mathrm{fl}_q(\theta_q(ab)), \quad \theta_q = 1 + 2\mathbf{u}_q \in \mathbb{F}_q \tag{28}$$

is satisfied from the standard floating-point arithmetic (Kearfott et al., 1994), and

$$1 + \frac{n\mathbf{u}_q}{1 - n\mathbf{u}_q} = \frac{1}{1 - n\mathbf{u}_q} \le \kappa_q, \quad \kappa_q := \mathrm{fl}_q\left(\frac{1}{1 - (n + 1)\mathbf{u}_q}\right), \quad (n + 1)\mathbf{u}_q < 1 \tag{29}$$

is obtained from (Ogita et al., 2005). For $0 \le a \in \mathbb{R}$, we have

$$a \le \mathrm{fl}_r(\theta_r \mathtt{RN}_r(a)). \tag{30}$$

For matrices $X = |X| \in \mathbb{F}_q^{m\times n}$ and $Y = |Y| \in \mathbb{F}_q^{n\times p}$, and the vectors in (24), XY is bounded by

$$XY \le \min\left(e(gY), (Xh)f^T\right) \le \mathrm{fl}_q\left(\theta_q \min\left(\mathrm{fl}_q\left(e(gY)\right), \mathrm{fl}_q\left((Xh)f^T\right)\right)\right) =: z \tag{31}$$

using only $\mathrm{fl}_q(\cdot)$ from (28). We define the function $\mathtt{fn2}(X, Y, q)$ producing z in (31).

For matrices $A \in \mathbb{F}_q^{m\times n}$ and $B \in \mathbb{F}_q^{n\times p}$, the paper (Rump, 2015) introduced

$$|A||B| \le \left(1 + \frac{n\mathbf{u}_q}{1 - n\mathbf{u}_q}\right) \mathrm{fl}_q(|A||B|). \tag{32}$$

To apply the error-free transformation for matrix multiplication to A_mB_m, we compute (5), (6) and (7) for A_m and B_m up to $k - 1$. Then, A_m and B_m are divided into k matrices, such that

$$\begin{aligned} A_m &= A_m^{(1)} + A_m^{(2)} + \ldots + \underline{A}_m^{(k)},\ A_m^{(i)} \in \mathbb{F}_r^{m\times n},\ 1 \le i \le k - 1, \quad A_m^{(k)} \in \mathbb{F}_q^{m\times n}, \\ B_m &= B_m^{(1)} + B_m^{(2)} + \ldots + \underline{B}_m^{(k)},\ B_m^{(j)} \in \mathbb{F}_r^{n\times p},\ 1 \le j \le k - 1, \quad B_m^{(k)} \in \mathbb{F}_q^{n\times p}. \end{aligned}$$

Let $\Delta_A \in \mathbb{F}_q^{m\times n}$ be $\Delta_A = \underline{A}_m^{(k)} - \mathtt{RN}_r\left(\underline{A}_m^{(k)}\right)$. Then, A_m is transformed into

$$A_m = A_m^{(1)} + A_m^{(2)} + \ldots + A_m^{(k-1)} + \mathtt{RN}_r\left(\underline{A}_m^{(k)}\right) + \Delta_A.$$

Now, we have

$$\langle A_m, A_r\rangle \subseteq \langle C_A,\ A_r + |\Delta_A|\rangle, \quad C_A \ = \ A_m - \Delta_A = A_m^{(1)} + A_m^{(2)} + \ldots + A_m^{(k-1)} + \mathtt{RN}_r\left(\underline{A}_m^{(k)}\right).$$

Let $A'_r = \mathrm{fl}_r(\theta_r\mathtt{RN}_r(\mathrm{fl}_q(\theta_q(A_r + |\Delta_A|))))$ and $B'_r = \mathrm{fl}_r(\theta_r\mathtt{RN}_r(B_r))$. Then, from (30) and (28), $A_r + |\Delta_A| \le A'_r$ and $B_r \le B'_r$ are satisfied, and we have $\langle A_m, A_r\rangle \subseteq \langle C_A,\ A'_r\rangle$ and $\langle B_m, B_r\rangle \subseteq \langle B_m,\ B'_r\rangle$. Therefore,

$$\langle A_m, A_r\rangle\langle B_m, B_r\rangle \subseteq \langle C_A, A'_r\rangle\langle B_m, B'_r\rangle \subseteq \langle C_AB_m,\ |C_A|B'_r + A'_r(|B_m| + B'_r)\rangle. \tag{33}$$

We focused on how to obtain enclosure of the center C_AB_m and the upper bound of the radius $|C_A|B'_r + A'_r(|B_m| + B'_r)$ in (33) by using only floating-point arithmetic with rounding to nearest mode.

First, we considered an upper bound of the radius in (33). Let $C'_A = \mathrm{fl}_r(\theta_r\mathtt{RN}_r(|C_A|))$ and $B'_m = \mathrm{fl}_r(\theta_r\mathtt{RN}_r(|B_m|))$. Using (28), (29) and (32) gives

$$|C_A|B'_r \le C'_AB'_r \le \left(1 + \frac{n\mathbf{u}_r}{1 - n\mathbf{u}_r}\right)\mathrm{fl}_r(C'_AB'_r) \le \kappa_r\mathrm{fl}_r(C'_AB'_r) \le \mathrm{fl}_r(\theta_r(\kappa(C'_AB'_r))) =: X_1,$$

and

$$\begin{aligned} A'_r(|B_m| + B'_r) \ &\le \ A'_r(B'_m + B'_r) \le \left(1 + \frac{n\mathbf{u}_r}{1 - n\mathbf{u}_r}\right)\mathrm{fl}_r(A'_rW) \\ &\le \ \kappa_r\mathrm{fl}_r(A'_rW) \le \mathrm{fl}_r(\theta_r(\kappa_r(A'_rW))) =: X_2, \end{aligned}$$

where $W = \mathrm{fl}_r(\theta_r(B'_m + B'_r))$. Finally, the radius in (33) is bounded by

$$|C_A|B'_r + A'_r(|B_m| + B'_r) \le \mathrm{fl}_q(\theta_q(X_1 + X_2)) =: Z.$$

Now, $\langle A_m,\ A_r\rangle \cdot \langle B_m,\ B_r\rangle \subseteq \langle C_AB_m,\ Z\rangle$. Note that it is possible to use the function `fn2` for the upper bound of $|C_A|B'_r + A'_r(|B_m| + B'_r)$.

Next, we treat the enclosure of C_AB_m in (33) that can be transformed into

$$C_AB_m = \sum_{i+j\le k} A_m^{(i)}B_m^{(j)} + \sum_{i=1}^{k-1} A_m^{(i)}\underline{B}_m^{(k-i+1)} + \mathtt{RN}_r(\underline{A}_m^{(k)})B_m. \tag{34}$$

There are $\dfrac{1}{2}k(k-1)$ matrix multiplications in (34).

We cannot apply low-precision arithmetic directly to (34), because

$$\underline{B}_m^{(k-i+1)} \notin \mathbb{F}_r^{m\times n}, \quad B_m \notin \mathbb{F}_r^{m\times n}$$

can be satisfied. Let

$$\underline{B}_m^{(k-i+1)} \ = \ \dot{B}_m^{(k-i+1)} + \Delta_{Bm}^{(k-i+1)}, \quad \dot{B}_m^{(k-i+1)} = \mathtt{RN}_r(B_m^{(k-i+1)}), \quad \Delta_{Bm}^{(k-i+1)} \in \mathbb{F}_q^{n\times p}, \tag{35}$$

$$B_m \ = \ \dot{B}_m + \Delta_{Bm}, \quad \dot{B}_m = \mathtt{RN}_r(B_m), \quad \Delta_{Bm} \in \mathbb{F}_q^{n\times p}. \tag{36}$$

From (1), we have

$$|\Delta_{Bm}^{(k-i+1)}| \le \mathbf{u}_r |\dot{B}_m^{(k-i+1)}|, \quad |\Delta_{Bm}| \le \mathbf{u}_r |\dot{B}_m|. \tag{37}$$

Note that

$$\mathrm{fl}_r\left(A_m^{(i)} B_m^{(j)}\right) = A_m^{(i)} B_m^{(j)}, \quad i+j \le k,$$

and it is possible that

$$\mathrm{fl}_r\left(A_m^{(i)} \dot{B}_m^{(k-i+1)}\right) \ne A_m^{(i)} \dot{B}_m^{(k-i+1)}, \quad 1 \le i \le k-1, \quad \mathrm{fl}_r\left(\mathtt{RN}_r(\underline{A}_m^{(k)})\dot{B}_m\right) \ne \mathtt{RN}_r(\underline{A}_m^{(k)})\dot{B}_m.$$

From (16), we have an enclosure of $A_m^{(i)} \dot{B}_m^{(k-i+1)}$ as

$$A_m^{(i)} \dot{B}_m^{(k-i+1)} \in \left\langle \mathrm{fl}_r\left(A_m^{(i)} \dot{B}_m^{(k-i+1)}\right), n\mathbf{u}_r |A_m^{(i)}||\dot{B}_m^{(k-i+1)}| \right\rangle.$$

Using (35), (16), (31), and (28), for $1 \le i \le k-1$ gives

$$\begin{aligned}
A_m^{(i)} \underline{B}_m^{(k-i+1)} &= A_m^{(i)}\left(\dot{B}_m^{(k-i+1)} + \Delta_{Bm}^{(k-i+1)}\right) \subseteq \left\langle A_m^{(i)} \dot{B}_m^{(k-i+1)},\ |A_m^{(i)}||\Delta_{Bm}^{(k-i+1)}| \right\rangle \\
&\subseteq \left\langle \mathrm{fl}_r\left(A_m^{(i)} \dot{B}_m^{(k-i+1)}\right),\ n\mathbf{u}_r |A_m^{(i)}||\dot{B}_m^{(k-i+1)}| + \mathbf{u}_r |A_m^{(i)}||\dot{B}_m^{(k-i+1)}| \right\rangle \\
&= \left\langle \mathrm{fl}_r\left(A_m^{(i)} \dot{B}_m^{(k-i+1)}\right),\ |A_m^{(i)}|\left((n+1)\mathbf{u}_r |\dot{B}_m^{(k-i+1)}|\right) \right\rangle \\
&\subseteq \left\langle \mathrm{fl}_r\left(A_m^{(i)} \dot{B}_m^{(k-i+1)}\right),\ |A_m^{(i)}|\left(\mathrm{fl}_q(\theta_q((n+1)\mathbf{u}_r |\dot{B}_m^{(k-i+1)}|)\right) \right\rangle \\
&\subseteq \left\langle C_m^{(i)},\ C_r^{(i)} \right\rangle
\end{aligned}$$

where

$$C_m^{(i)} = \mathrm{fl}_r\left(A_m^{(i)} \dot{B}_m^{(k-i+1)}\right), \quad C_r^{(i)} = \mathtt{fn2}\left(|A_m^{(i)}|,\ \left(\mathrm{fl}_q(\theta_q((n+1)\mathbf{u}_r |\dot{B}_m^{(k-i+1)}|)\right),\ q\right)$$

for $1 \le i \le k-1$. Similarly, from (36), (16), (31), and (28), we obtain

$$\begin{aligned}
\mathtt{RN}_r(\underline{A}_m^{(k)}) B_m &= \mathtt{RN}_r(\underline{A}_m^{(k)})(\dot{B}_m + \Delta_{Bm}) \subseteq \left\langle \mathtt{RN}_r(\underline{A}_m^{(k)})\dot{B}_m,\ |\mathtt{RN}_r(\underline{A}_m^{(k)})||\Delta_{Bm}| \right\rangle \\
&\subseteq \left\langle \mathrm{fl}_r\left(\mathtt{RN}_r(\underline{A}_m^{(k)})\dot{B}_m\right),\ n\mathbf{u}_r |\mathtt{RN}_r(\underline{A}_m^{(k)})||\dot{B}_m| + \mathbf{u}_r |\dot{A}_m^{(k)}||\dot{B}_m| \right\rangle \\
&= \left\langle \mathrm{fl}_r\left(\mathtt{RN}_r(\underline{A}_m^{(k)})\dot{B}_m\right),\ |\mathtt{RN}_r(\underline{A}_m^{(k)})|((n+1)\mathbf{u}_r |\dot{B}_m|) \right\rangle \\
&\subseteq \left\langle \mathrm{fl}_r\left(\mathtt{RN}_r(\underline{A}_m^{(k)})\dot{B}_m\right),\ |\mathtt{RN}_r(\underline{A}_m^{(k)})|\mathrm{fl}_q(\theta_q((n+1)\mathbf{u}_r |\dot{B}_m|)) \right\rangle \\
&= \langle C_m^{(k)},\ C_r^{(k)} \rangle
\end{aligned}$$

where

$$C_m^{(k)} = \mathrm{fl}_r\left(\mathtt{RN}_r(\underline{A}_m^{(k)})\dot{B}_m\right), \quad C_r^{(k)} = \mathtt{fn2}\left(|\mathtt{RN}_r(\underline{A}_m^{(k)})|,\ \mathrm{fl}_q(\theta_q((n+1)\mathbf{u}_r |\dot{B}_m|)),\ q\right)$$

Until now, we have

$$C_A C_B \in \left\langle \sum_{i+j\le k} \mathrm{fl}_r\left(A_m^{(i)} B_m^{(j)}\right),\ \mathbf{O} \right\rangle + \sum_{i=1}^{k} \langle C_m^{(i)},\ C_r^{(i)} \rangle$$
$$= \left\langle \sum_{i=1}^{\frac{1}{2}k(k-1)} D^{(i)} + \sum_{i=1}^{k} C_m^{(i)},\ \sum_{i=1}^{k} C_r^{(i)} \right\rangle,$$

where $D^{(1)} := \mathrm{fl}_r(A^{(1)}B^{(1)})$, $D^{(2)} := \mathrm{fl}_r(A^{(1)}B^{(2)})$, $D^{(3)} := \mathrm{fl}_r(A^{(2)}B^{(1)}), \ldots, D^{(\frac{1}{2}k(k-1))} := \mathrm{fl}_r(A^{(k-1)}B^{(1)})$. Let $F^{(1)} := D^{(1)}$, $G^{(1)} := \mathbf{O}$ and $H^{(1)} := C_r^{(1)}$. Compute

$$F^{(i+1)} := \mathrm{fl}_q\left(F^{(1)} + D^{(i+1)}\right), \quad G^{(i+1)} := \mathrm{fl}_q\left(\theta_q\left(G^{(i)} + \mathbf{u}_q|F_{(i+1)}|\right)\right),$$

for $i = 2, 3, \ldots, \frac{1}{2}(k^2 - k) - 1$ and

$$H^{(j+1)} := \mathrm{fl}_q\left(\theta_q\left(C_r^{(j+1)} + H^{(j)}\right)\right)$$

for $j = 1, \ldots, k-1$. $C_A B_m$ is enclosed by

$$C_A B_m \in \left\langle F^{(\frac{1}{2}(k^2-k))}, Q \right\rangle, \quad Q := \mathrm{fl}_q(\theta_q(G^{(\frac{1}{2}(k^2-k))} + H^{(k)})),$$

and we finally obtain

$$\langle A_m,\ A_r\rangle\langle B_m,\ B_r\rangle \in \langle F^{(\frac{1}{2}k(k-1))},\ \mathrm{fl}_q(\theta_q(Q + Z))\rangle. \tag{38}$$

Setting $q = 64$ and $r = 32$ derives

$$PA^{(i)} \in \mathbb{F}_{16}^{m\times n}, \quad B^{(j)}Q \in \mathbb{F}_{16}^{n\times p}$$

are satisfied where P, Q are diagonal matrices for proper scaling. Because Tensor Cores accepts binary16 format and compute it as binary32, no rounding error occurs in the evaluation of $A^{(i)}B^{(j)}$, $i + j \le k$ using Tensor Cores (Mukunoki et al., 2020). Performance is accelerated when the size of the matrices is sufficiently large. This will be shown in the next section.

5. Numerical Examples

In this section, we present numerical examples to illustrate the efficiency of the proposed methods. We used the following components: CPU: Core i7-8665U, GPU: Quadro RTX 5000, OS: Windows 10, Software: MATLAB R2020a with Parallel Computing Toolbox and CUDA version 10.0.130. First, we determined the difference in performance between binary32 arithmetic and binary64 arithmetic for both the CPU and GPU. Tables I and II show the computing time of a product of square matrices using MATLAB. Each item shows an average of 20 examples. Note that computing time

in this section did not include the transfer time of data between CPU and GPU. The number in parentheses is represented in the form: precision (bit), e.g., CPU (32) means binary32 on the CPU. GPU (16-32) means computation using Tensor Cores with cublasGemmEx in cuBLAS. The inputs in cublasGemmEx were binary16, and the output was binary32 in the numerical examples. Note that scaling for matrices was necessary for converting binary32 to binary16 in many cases: otherwise, overflow or underflow tended to occur due to a narrow range of the exponent of binary16. Hence, GPU (16-32) included the time for diagonal scaling. The following is the detail of the benchmark using MATLAB.

- CPU (32) and CPU (64): $A * B$ for randomly generated matrices A and B
- GPU (32) and GPU (64): $A * B$ (after $A = \mathrm{gpuArray}(A)$; and $B = \mathrm{gpuArray}(B)$;, the time for gpuArray is not included)
- GPU (16-32): transforming the matrices with binary32 into matrices with binary16 and calling cublasGemmEx in cuBLAS exploiting MATLAB executable. The code is compiled using mexcuda.

If the matrices' dimensions were over 2,000, the ratio GPU (64) / GPU (32) was more than 15. There is an advantage to using binary32 on the GPU. For example, among NVIDIA GPUs after the Maxwell architecture (released in 2014), GeForce, Tegra, and some low-cost Tesla products have binary64 performance less than 1/32 of binary32, and thus the proposed method will be effective.

However, because the ratio CPU (64) / CPU (32) was approximately 2, there was no merit in using binary32 on the CPU for this problem.

Table I. Computing times (sec) for matrix multiplication (CPU).

Dimension	CPU (32)	CPU (64)	ratio (64/32)
2,000	0.086	0.1592	1.85
4,000	0.5090	1.1211	2.20
6,000	1.8083	3.9009	2.15
8,000	4.3411	9.3109	2.14
10,000	8.1339	17.7706	2.18
12,000	13.3200	31.8232	2.38

Interval matrices $\langle A_m, A_r \rangle$ and $\langle B_m, B_r \rangle$ were generated using MATLAB2020a as follows

$$\begin{aligned} A_m &= \mathtt{randn}(n); \quad A_r = c_1 * \mathtt{rand}(n). * \mathtt{abs}(A_m); \\ B_m &= \mathtt{randn}(n); \quad B_r = c_2 * \mathtt{rand}(n). * \mathtt{abs}(B_m); \end{aligned}$$

Table II. Computing times (sec) for matrix multiplication (GPU).

Dimension	GPU (16-32)	GPU (32)	GPU (64)	ratio (64/(16-32))	ratio (64/32)
2,000	0.0049	0.0032	0.0524	10.7937	16.4890
4,000	0.0143	0.0225	0.3987	27.7866	17.7414
6,000	0.0485	0.0749	1.3573	27.9910	18.1275
8,000	0.0672	0.1932	3.2473	48.3451	16.8092
10,000	0.1581	0.3890	6.4355	40.7028	16.5422
12,000	0.3577	0.6031	11.0075	30.7695	18.2517

where c_1 and c_2 are positive constants and the width of the interval can be controlled by the constants c_1 and c_2.

Table III shows the average radii of the computed intervals. The notation, MD, M0, and M1 to M7, in the tables in this section indicates

- MD: The method (Rump, 1999) using CPU, as reference.
- M0: The method based on (20) using GPU (binary64 and binary32)
- Mk: The proposed method based on (38) using GPU, $k = 2, 3, ..., 7$

The method MD involves two matrix multiplications with double-precision and two matrix multiplications with single-precision:

$$\mathrm{fl}_{\nabla,64}(A_m B_m), \quad \mathrm{fl}_{\triangle,64}(A_m B_m), \quad \mathrm{fl}_{\triangle,32}(A'_m B'_r), \quad \mathrm{fl}_{\triangle,32}(A'_r S),$$

where $S := \mathtt{fl}((1 + 2u_{32})(B'_m + B'_r))$. The method M0 involves one matrix multiplication with double-precision and two matrix multiplications with single-precision:

$$\mathrm{fl}_{64}(A_m B_m), \quad \mathrm{fl}_{32}(A'_m P), \quad \mathrm{fl}_{32}(A'_r Q).$$

where $P, Q \in \mathbb{F}^{n\times p}$ are the upper bounds of $n\mathbf{u}_q B'_m + B'_r$ and $B'_m + B_r$, respectively. The method Mk has no matrix multiplication with double-precision and $\frac{1}{2}k(k-1) + 2$ matrix multiplications with single-precision.

Computing times for each method with and without Tensor Cores are shown in Table V. The time is an average of 10 examples. We underline comparable methods to M0. Notation TC indicates that cublasGemmEx in cuBLAS was used for $A^{(i)}B^{(j)}$, $i + j \leq k$. The ratio of computing time, the time without Tensor Cores is normalized to be one, is shown as AC in Table V. If the matrix dimension was 1,000, then using Tensor Cores slowed down performance because the cost of the scaling is not negligible. If the dimension was more than 3,000, we observed an acceleration of performance using Tensor Cores.

A summary of the numerical example from the tables is as follows:

Table III. Average of the radii ($n = 7,000,\ c_1 = 10^{-15}$).

c_2	MD	M0	M2	M3	M4	M5	M6	M7
10^{-15}	8.68e-12	3.47e-09	5.82e-01	2.19e-02	5.44e-04	1.16e-05	2.31e-07	4.38e-09
10^{-12}	2.23e-09	5.69e-09	5.82e-01	2.19e-02	5.43e-04	1.16e-05	2.32e-07	6.59e-09
10^{-9}	2.22e-06	2.23e-06	5.82e-01	2.18e-02	5.43e-04	1.38e-05	2.45e-06	2.23e-06
10^{-6}	2.22e-03	2.22e-03	5.84e-01	2.41e-02	2.77e-03	2.24e-03	2.22e-03	2.22e-03

Table IV. Average of the radii ($n = 7,000,\ c_1 = c_2$).

$c_1,\ c_2$	MD	M0	M2	M3	M4	M5	M6	M7
10^{-15}	8.68e-12	3.47e-09	5.82e-01	2.19e-02	5.44e-04	1.16e-05	2.31e-07	4.38e-09
10^{-12}	4.45e-09	7.92e-09	5.82e-01	2.19e-02	5.43e-04	1.16e-05	2.35e-07	8.83e-09
10^{-9}	4.45e-06	4.46e-06	5.82e-01	2.19e-02	5.47e-04	1.60e-05	4.68e-06	4.46e-06
10^{-6}	4.45e-03	4.45e-03	5.86e-01	2.64e-02	5.00e-03	4.46e-03	4.45e-03	4.45e-03

$c_2 = 1e-15$ and $c_2 = 1e-12$: The tightness of the interval by M0 and M7 was comparable. M7 was slightly slower than M0.

$c_2 = 1e-09$: The tightness of the interval by M0 and M6 was comparable. M6 was slightly faster than M0.

$c_2 = 1e-06$: The tightness of the interval by M0 and M5 was comparable. M5 was much faster than M0.

If the radius was sufficiently small compared with the magnitude of the center, the original method M0 was better than the proposed methods. Otherwise, the proposed methods were faster than M0 whereas tightness of the produced intervals was comparable.

Conclusion

We proposed methods for interval matrix multiplication using low-precision arithmetic based on error-free transformation. Tensor Cores efficiently accelerated the performance of interval matrix multiplication when matrix dimension was more than 3,000. The numerical examples illustrated that the proposed methods have an advantage of higher performance compared with those from previous works. Exploiting batched BLAS will be our future work for small matrices.

Table V. Computing time (sec) with and without Tensor Cores.

Dimension		MD	M0	M2	M3	M4	M5	M6	M7
		0.053	0.028	0.018	0.022	0.026	0.030	0.034	0.039
1000	TC	-	-	0.018	0.025	0.032	0.042	0.053	0.065
	AC	-	-	0.984	1.121	1.240	1.391	1.557	1.6523
		0.785	0.296	0.151	0.194	0.253	0.318	0.395	0.479
3000	TC	-	-	0.142	0.182	0.226	0.275	0.340	0.422
	AC	-	-	0.946	0.934	0.894	0.865	0.861	0.880
		3.580	1.190	0.542	0.698	0.906	1.152	1.476	1.802
5000	TC	-	-	0.495	0.627	0.787	0.944	1.178	1.452
	AC	-	-	0.913	0.899	0.868	0.819	0.798	0.805
		9.417	3.011	1.224	1.662	2.171	2.855	3.561	4.426
7000	TC	-	-	1.127	1.533	1.906	2.333	2.947	3.694
	AC	-	-	0.921	0.922	0.877	0.816	0.827	0.834
		20.139	6.097	2.466	3.316	4.479	5.879	7.570	9.509
9000	TC	-	-	2.219	2.779	3.481	4.147	4.983	6.211
	AC	-	-	0.899	0.838	0.777	0.705	0.658	0.653

Acknowledgements

This work was supported by JST CREST Grant Number JPMJCR14D4, Japan.

References

ANSI/IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic, New York, 2008.

Basic Linear Algebra on NVIDIA GPUs, https://developer.nvidia.com/cublas.

Demmel J. and Hida Y. Accurate and efficient floating point summation. *SIAM Journal on Scientific Computing*, 25(4) , 1214–1248, 2004.

Frommer, A. and Hashemi, B. Verified computation of square roots of a matrix. *SIAM Journal on Matrix Analysis and Applications*, SIAM, 31(3):1279–1302, 2010.

Frommer A. and Hashemi B. Verified error bounds for solutions of Sylvester matrix equations. *Linear Algebra and its Applications*, 436(2):405–420, 2012.

Hoshi T., Ogita T., Ozaki K. and Terao T. An a posteriori verification method for generalized real-symmetric eigenvalue problems in large-scale electronic state calculations. *Journal of Computational and Applied Mathematics*, 376:112830, 2020.

Jeannerod, C.-P. and Rump S. M. Improved error bounds for inner products in floating-point arithmetic. *SIAM Journal on Matrix Analysis and Applications*, 34(2):338–334, 2013.

Kazal N.Y., Mukhlash I.,Arry S.B., Hidayat N. and Ozaki K. Extended use of error-free transformation for real matrix multiplication to complex matrix multiplication. *Journal of Physics: Conference Series*, to appear.

Kearfott, R. B., et al. Algorithm 737: INTLIB: a portable Fortran 77 interval standard-function library. ACM Transactions on Mathematical Software (TOMS), 20(4):447–459, 1994.

Minamihata A., Ozaki K., Ogita T. and Oishi S. Improved Extraction Scheme for Accurate Floating-point Summation. The 35th JSST Annual Conference, International Conference on Simulation Technology (JSST2016), Kyoto University (2016/10/27-29).

Miyajima S., Ogita T., Rump S.M. and Oishi S. Fast verification for all eigenpairs in symmetric positive definite generalized eigenvalue problem. *Reliable Computing*, 14:24–45, 2010.

Miyajima S. Numerical enclosure for each eigenvalue in generalized eigenvalue problem. *Journal of Computational and Applied Mathematics*, 236(9):2545–2552, 2012.

Miyajima S. Fast enclosure for solutions of Sylvester equations. *Linear Algebra and its Applications*, 439(4):856–878, 2013.

Moore, R. E. *Interval analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

Mukunoki D., Ozaki K., Ogita T. and Imamura T. DGEMM Using Tensor Cores, and Its Accurate and Reproducible Versions. In: Sadayappan P., Chamberlain B., Juckeland G., Ltaief H. (eds) High Performance Computing. ISC High Performance 2020. Lecture Notes in Computer Science, 12151:230–248, 2020.

Ogita, T. and Oishi, S. Fast Inclusion of Interval Matrix Multiplication. *Reliable Computing*, 11(3):191–205, 2005.

Ogita T., Rump S.M. and Oishi S. Verified solution of linear systems without directed rounding. Technical Report 2005-04, Advanced Research Institute for Science and Engineering, Waseda University, 2005.

Ogita T., Rump S.M. and Oishi S. Accurate sum and dot product. *SIAM Journal on Scientific Computing*, 26(6), 1955–1988, 2005.

Oishi S. Fast enclosure of matrix eigenvalues and singular values via rounding mode controlled computation. *Linear Algebra and its Applications*, 324(1–3): 133–146, 2001.

Ozaki, K., Ogita, T., Bünger. F. and Oishi, S. Accelerating interval matrix multiplication by mixed precision arithmetic. *Nonlinear Theory and its Applications*, 6(3):364–376, 2015.

Ozaki, K., Ogita, T. and Oishi, S. Error-free Transformation of Matrix Multiplication with a Posteriori Validation. *Numerical Linear Algebra with Applications*, 23(5):931–946, 2016.

Ozaki, K., Ogita, T., Oishi, S. and Rump S. M. Fast Algorithms for Floating-point Interval Matrix Multiplication. *Journal of Computational and Applied Mathematics*, 236:1795–1814, 2012.

Ozaki, K., Ogita, T., Oishi, S. and Rump S. M. Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications. *Numerical Algorithms*, 59(1):95–118, 2012.

Rump S. M. Fast and parallel interval arithmetic. *BIT Numerical Mathematics*, 39(3):539–560, 1999.

Rump S.M. Computer-assisted proofs and Self-Validating Methods. In B. Einarsson, editor, Handbook on Accuracy and Reliability in Scientific Computation, SIAM, 195-240, 2005.

Rump S.M. Ultimately fast accurate summation. *SIAM Journal on Scientific Computing*, 31(5), 3466–3502, 2009.

Rump S. M. Fast Interval Matrix Multiplication. *Numerical Algorithms*, 61(1):1–34, 2012.

Rump S. M. Computable backward error bounds for basic algorithms in linear algebra. *Nonlinear Theory and its Applications*, 6(3):360–363, 2015.

Rump, S. M., Ogita, T. and Oishi S. Accurate floating-point summation part I: Faithful rounding. *SIAM Journal on Scientific Computing*, 31(1):189–224, 2008.

Rump, S.M., Ogita T. and Oishi S. Accurate floating-point summation part II: sign, K-kold faithful and rounding to nearest. *SIAM Journal on Scientific Computing*, 31 (2), 1269–1302, 2008.

Strassen V. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

Sunaga, T. Theory of interval algebra and its application to numerical analysis. *RAAG memoirs*, 2:29–46, 1958.

Winograd S. On multiplication of 2 x 2 matrices. *Linear Algebra and its Applications*, 4:381–388, 1971.

Yamanaka N., Ogita T., Rump S.M. and Oishi S. A parallel algorithm of accurate dot product. *Parallel Computing*, 34 (6–8), 392–410, 2008.

Zhu Y.-K. and Hayes W.B. Correct rounding and a hybrid approach to exact floating-point summation. *SIAM Journal on Scientific Computing*, 31 (4), 2981–3001, 2009.

Zhu, Y.-K. and Hayes W.B. Algorithm 908: online exact summation of floating-point streams. *ACM Transactions on Mathematical Software*, 37 (3), 37:1–37:13, 2010.