

Ozaki Scheme II: A GEMM-oriented emulation of floating-point matrix multiplication using an integer modular technique

Katsuhisa Ozaki¹, Yuki Uchino², and Toshiyuki Imamura²

Journal Title
 XX(X):1–13
 ©The Author(s) 2016
 Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
 DOI: 10.1177/ToBeAssigned
www.sagepub.com/



Abstract

This paper addresses emulation algorithms for matrix multiplication. General Matrix-Matrix Multiplication (GEMM), a fundamental operation in the Basic Linear Algebra Subprograms (BLAS), is typically optimized for specific hardware architectures. The Ozaki scheme is a well-established GEMM-based emulation method for matrix multiplication, wherein input matrices are decomposed into several low-precision components to ensure that the resulting matrix product is computed exactly through numerical operations. This study proposes a novel GEMM-based emulation method for matrix multiplication that leverages the Chinese Remainder Theorem. The proposed method inherits the computational efficiency of highly optimized GEMM routines and further enables control over the number of matrix multiplications, which can enhance computational accuracy. We present numerical experiments featuring INT8 Tensor Core operations on GPUs and FP64 arithmetic on CPUs as case studies. The results demonstrate that FP64 emulation using the proposed method achieves performance levels of up to 7.4 to 9.8 TFLOPS on the NVIDIA RTX 4090 and 56.6 to 80.2 TFLOPS on the NVIDIA GH200, exceeding the measured performance of native FP64 arithmetic. Furthermore, for FP64 computations on CPUs, the proposed method achieved up to a 2.3x speedup in emulating quadruple-precision arithmetic compared to the conventional Ozaki scheme.

Keywords

matrix multiplication, floating-point arithmetic, matrix engines, high-precision emulation

1 Introduction

This paper discusses numerical methods for emulating high-precision matrix multiplication. When the accuracy of the computed results is insufficient, multiple-precision arithmetic provides a viable alternative (Higham 2018). For such computations, high-precision datatypes, such as the built-in `float128_t` in the C++23 standard, and multiple-precision arithmetic libraries, such as MPFR (Fousse et al. 2025), offer robust and reliable functionality. In the domain of linear algebra, LAPACK (Nakata 2022) is available for high-precision problem solving. Alternatively, when it is unnecessary to extend the exponent range of floating-point numbers and only the significand requires pseudo-extension, *multiple-component arithmetic* provides an efficient solution. Such methods include double- and quad-word arithmetic (Hida et al. 2001; Bailey et al. 2002), as well as triple-word arithmetic (Muller et al. 2015).

When the computational task is limited to matrix multiplication, the Ozaki scheme (Ozaki et al. 2012, 2013) is recognized as a highly reliable method. It achieves high accuracy by leveraging standard floating-point operations and high performance by exploiting optimized routines, e.g., General Matrix-Matrix Multiplication (GEMM), in Basic Linear Algebra Subprograms (BLAS).

In recent years, increasing attention has been paid to matrix engines optimized for low-precision arithmetic. From the perspective of power efficiency, mixed-precision computation utilizing low-precision formats has also

¹Department of Mathematical Sciences, Shibaura Institute of Technology, Japan

²RIKEN Center for Computational Science, Japan

Corresponding author:

Katsuhisa Ozaki, 307 Fukasaku, Minuma-ku, Saitama-shi, Saitama 337-8570, Japan

Email: ozaki@sic.shibaura-it.ac.jp

garnered significant interest. Table 1 presents the floating-point and integer performance at various precisions on NVIDIA GPUs. Note that the specification of FP16 Tensor Cores (TCs) on RTX 4090 (165.2 TFLOPS) is for FP16 input and FP32 output, the specifications for the H100 and GH200 are the same as those for the H200. While FP64 performance shows a substantial gap between data-center-class and consumer-grade GPUs, FP16 and INT8 TCs operations offer outstanding throughput.

Table 1. Specifications in TFLOPS/TOPS of GPUs (NVIDIA Corporation 2024) for dense data

	Ampere A100 SXM4	ADA RTX 4090	Hopper H200 SXM5	Blackwell B200
	GA100	AD102-300		
FP64	9.7	1.29	34	40
FP64 TC	19.5	—	67	40
FP32	19.5	82.6	67	80
TF32 TC	156	82.6	494	1100
BF16 TC	312	165.2	989	2250
FP16 TC	312	165.2	989	2250
INT8 TC	624	660.6	1979	4500
FP8 TC	—	660.6	1979	4500
FP6 TC	—	—	—	4500
INT4 TC	—	1321.2	—	—
FP4 TC	—	—	—	9000

The use of FP16 TCs in the Ozaki scheme has been discussed (Mukunoki et al. 2020). Subsequently, Ootomo et al. (2024) employed the Ozaki scheme with INT8 TCs to emulate matrix multiplication in FP64, and Ootomo (2024) released the API `ozIMMU` as open source. Uchino et al. (2025) further accelerated `ozIMMU`, and the enhanced version is also publicly released (Uchino 2024). The potential applications of the Ozaki scheme are explored (Dawson et al. 2024), and its integration into the High-Performance Linpack (HPL) benchmark is demonstrated (Dongarra et al. 2024).

In this study, we propose a novel matrix multiplication emulation method based on the Chinese Remainder Theorem. Similarly to emulation methods based on the conventional Ozaki scheme, the proposed approach is also GEMM-based, allowing the use of INT8 matrix engines. In contrast to the conventional Ozaki scheme, the proposed method significantly reduces the number of required matrix multiplications, which is a notable advantage. While the conventional Ozaki scheme controls accuracy by adjusting the number of slices, the proposed method enables precise control over the number of matrix multiplications.

We present numerical results obtained using INT8 TCs on the NVIDIA GH200 Grace Hopper Superchip and the NVIDIA GeForce RTX 4090 GPU, as well as FP64 operations on the Intel® Core™ i7-8665U processor and the

Intel® Core™ i9-10980XE processor. The proposed method achieved 56.6–80.2 TFLOPS in FP64-equivalent precision on the GH200, and 7.4–9.8 TFLOPS on the RTX 4090. Given that the measured performance of FP64 TCs is 61.9 TFLOPS, the emulation achieves a performance that is comparable to or even exceeds native FP64. In addition, for quad-word arithmetic emulation on a CPU, the proposed method achieved an approximate 2.3x speedup compared to the conventional Ozaki scheme.

The remainder of this paper is organized as follows. Section 2 introduces the notation, the conventional Ozaki scheme, and the Chinese Remainder Theorem. Section 3 describes the proposed method (referred to as Ozaki scheme II) and outlines algorithms using INT8 TCs on GPU and FP64 arithmetic on CPU. Section 4 presents numerical experiments conducted on GPUs and CPUs to evaluate the effectiveness of the proposed method. Finally, Section 5 concludes the paper.

2 Notation and Previous Study

2.1 Notation

Let \mathbb{F} be a set of binary floating-point numbers as defined by IEEE Computer Society (2019). We define a constant u as the unit roundoff, e.g., $u = 2^{-53}$ for FP64. Let \mathbb{Z}_k for $k \in \mathbb{N}$ be a set of integers, where $a \in \mathbb{Z}_k$ means $|a| \leq 2^k$. The notation $f1(\cdot)$ indicates a computed result using floating-point arithmetic. We assume that neither overflow nor underflow occurs in $f1(\cdot)$. For a matrix $A = (a_{ij})$, the notation $|A|$ represents the matrix whose entry (i, j) is $|a_{ij}|$; that is, $|A| = (|a_{ij}|)$. The notation $\gcd(a, b)$ stands for the greatest common divisor of two integers a and b .

The modulo operation $a \bmod m$, where $a \in \mathbb{Z}$ and $m \in \mathbb{N}$, can be defined in various ways. One such variant is the symmetric modulo, in which the remainder r satisfies

$$-\frac{m}{2} \leq r \leq \frac{m}{2}.$$

In this paper, we adopt this symmetric definition. Formally, $r = a \bmod m$ is defined by

$$r = a - m \cdot \left\lfloor \frac{a}{m} + \frac{1}{2} \right\rfloor. \quad (1)$$

This definition ensures that the remainder is the integer closest to zero among those congruent to a modulo m . For a matrix A , the expression $A \bmod m$ refers to applying the modulo operation to each element of the matrix. This results in a new matrix of the same dimensions as A .

2.2 Conventional Ozaki scheme

For $A \in \mathbb{F}^{p \times q}$ and $B \in \mathbb{F}^{q \times r}$, our aim is to obtain an approximation of AB . The Ozaki scheme (Ozaki et al. 2012, 2013) splits the matrices into

$$\begin{aligned} A &= A_1 + A_2 + \cdots + A_{k-1} + \underline{A}_k, \\ B &= B_1 + B_2 + \cdots + B_{\ell-1} + \underline{B}_{\ell}, \end{aligned} \quad (2)$$

where

$$A_i \in \mathbb{F}^{p \times q}, \quad B_j \in \mathbb{F}^{q \times r}$$

for $1 \leq i \leq k-1$ and $1 \leq j \leq \ell-1$, and

$$\underline{A}_k := A - \sum_{i=1}^{k-1} A_i \in \mathbb{F}^{p \times q}, \quad \underline{B}_{\ell} := B - \sum_{i=1}^{\ell-1} B_i \in \mathbb{F}^{q \times r}.$$

For (2), we call k the number of slices for A and ℓ that for B . Here, we set $k = \ell$ according to Ozaki et al. (2012), but it is also possible to set $k \neq \ell$ as discussed by Ozaki et al. (2013).

Then, AB is transformed into

$$AB = \sum_{i+j \leq k} A_i B_j + \sum_{i=1}^{k-1} A_i \underline{B}_{k+1-i} + \underline{A}_k B. \quad (3)$$

The matrix products $A_i B_j$ for $i+j \leq k$ can be computed without rounding errors using floating-point arithmetic. Note that Ozaki et al. (2025) also proposed an alternative form of AB , but this is not considered in this paper.

The Ozaki scheme for $k = \ell$ consists of the following three parts:

Part 1: splitting matrices as in (2)

Part 2: computation of $k(k+1)/2$ matrix products as in (3)

Part 3: reduction of the computed matrix products

The computational costs are $\mathcal{O}(pq) + \mathcal{O}(qr)$ for Part 1, $k(k+1)pqr + \mathcal{O}(pr)$ for Part 2, and $\mathcal{O}(pr)$ for Part 3. Thus, Part 2 dominates the overall cost for sufficiently large p , q , and r . One advantage of the Ozaki scheme is that optimized BLAS routines can be applied to this computationally intensive part. However, a disadvantage is that a large amount of memory is required to store the matrices, as they are decomposed into multiple summands in (2). In Part 2, an appropriate BLAS routine is selected based on the structure of the matrices. For example, if A is a triangular matrix and B is a general matrix, TRMM is used. If A and B have no special structure, GEMM is used.

Even when A and B are represented as multi-component formats, a similar approach can achieve high-precision computational results. For example, let A and B be represented as double-word formats: $A := A_h + A_{\ell}$ and

$B := B_h + B_{\ell}$ for $A_h, A_{\ell} \in \mathbb{F}^{p \times q}$ and $B_h, B_{\ell} \in \mathbb{F}^{q \times r}$ such that

$$\text{fl}(A_h + A_{\ell}) = A_h, \quad \text{fl}(B_h + B_{\ell}) = B_h.$$

Similarly, we divide $A_h + A_{\ell}$ and $B_h + B_{\ell}$ into the unevaluated sum of floating-point matrices such that

$$\begin{aligned} A_h + A_{\ell} &\approx A_1 + A_2 + \cdots + A_{e-1} + \underline{A}_e, \\ B_h + B_{\ell} &\approx B_1 + B_2 + \cdots + B_{f-1} + \underline{B}_f, \end{aligned} \quad (4)$$

where $A_i \in \mathbb{F}^{p \times q}$, $B_j \in \mathbb{F}^{q \times r}$, and $\text{fl}(A_i B_j) = A_i B_j$ for $1 \leq i \leq e-1$ and $1 \leq j \leq f-1$. In addition,

$$\begin{aligned} \mathbb{F}^{p \times q} \ni \underline{A}_e &\approx A_h + A_{\ell} - \sum_{i=1}^{e-1} A_i, \\ \mathbb{F}^{q \times r} \ni \underline{B}_f &\approx B_h + B_{\ell} - \sum_{i=1}^{f-1} B_i. \end{aligned}$$

Then, we obtain a computed result based on (3). Note that in the calculations of Part 3, high-precision computations such as double-word arithmetic are required in this case.

Next, we introduce a method for emulating matrix multiplication using matrix engines available in recent GPUs. Typical examples include NVIDIA TCs. Let nonsingular diagonal matrices $D_i \in \mathbb{F}^{p \times p}$ and $E_i \in \mathbb{F}^{r \times r}$ whose diagonal elements are powers of two. This aim is to achieve an error-free diagonal scaling. Mukunoki et al. (2020) set

$$\begin{aligned} A &\approx D_1^{-1} D_1 A_1 + D_2^{-1} D_2 A_2 + \cdots + D_k^{-1} D_k A_k, \\ B &\approx B_1 E_1 E_1^{-1} + B_2 E_2 E_2^{-1} + \cdots + B_k E_k E_k^{-1}, \end{aligned} \quad (5)$$

where all elements in $D_i A_i$ and $B_i E_i$ for all $1 \leq i \leq k$ can be represented in FP16. The entries of the products $D_i A_i$ and $B_i E_i$ behave as though they possess a significand with

$$\left\lceil \frac{24 - \log_2 n}{2} \right\rceil \quad (6)$$

bits of precision. Then, $(D_i A_i)(B_j E_j)$ can be computed without rounding error using FP16 TCs. Note that FP16 TCs can store the results in either FP16 or FP32, but in this case, the computed results are stored in FP32. Then, the approximation \tilde{C} is obtained by

$$\tilde{C} := \text{fl} \left(\sum_{i+j \leq k+1} D_i^{-1} ((D_i A_i)(B_j E_j)) E_j^{-1} \right).$$

This involves $k(k+1)/2$ matrix multiplications. After computing matrix multiplication $(D_i A_i)(B_j E_j)$, the results

are converted from FP32 to FP64, and a summation of these is performed in FP64.

Otomo et al. (2024) similarly used the form (5) where all elements of $D_i A_i$ and $B_i E_i$ for all i are stored in INT8. Taking into account that the results of INT8 TCs operations are stored in INT32, they utilized this property to achieve error-free computation of the product $(D_i A_i)(B_j E_j)$ when $q \leq 2^{29}$ using INT8 TCs. The first advantage of using INT8 TCs is that it is theoretically twice or 4x as fast as FP16 TCs as in Table 1. The second advantage of using INT8 TCs is that if $q < 2^{17}$, the significand of the split matrices is kept 7-bit independent of q . If $q > 2^{10}$, the quantity (6) is less than seven.

The acceleration of ozIMMU and the analysis of rounding errors are discussed by Uchino et al. (2025). They achieved acceleration by reducing the cost of the reduction in Part 3. Specifically, they reduced the number of summation calculations performed using FP64. In this paper, we call this type of algorithm *Ozaki scheme I*. Ozaki scheme I using INT8 TCs consists of the following three parts:

Part 1: splitting matrices as in (5)

Part 2: computation of $k(k+1)/2$ matrix products using INT8 TCs

Part 3: reduction of the computed matrix products in FP64.

Here, we introduce the accuracy trends of Ozaki scheme I using INT8 TCs. In this case, $D_i A_i$ and $B_i E_i$ for all i in (5) are represented by INT8. We generated two matrices using MATLAB as

$$A = \text{randn}(1000), \quad B = \text{randn}(1000), \quad (7)$$

where $\text{randn}(n)$ generates an $n \times n$ matrix of normally distributed random numbers with mean 0 and variance 1. Table 2 shows the number of matrix multiplications required for each slice. Figure 1 shows the maximum relative error of the result computed by Ozaki scheme I for (7) for the number of slices (left) and the matrix multiplications (right). From Fig. 1, Ozaki scheme I improves in accuracy in proportion to the number of slices. When increasing the number of slices from k to $k+1$, $k+1$ additional matrix multiplications are required. Therefore, as shown in Fig. 1, the accuracy does not exhibit linear growth with respect to the number of matrix multiplications when plotted on a logarithmic scale.

2.3 Chinese Remainder Theorem

A brief introduction to the Chinese Remainder Theorem is provided, as it is essential for the proposed method.

Table 2. Relation between the number of slices and the number of matrix multiplications

slices	2	3	4	5	6	7	8	9	10
muls.	3	6	10	15	21	28	36	45	55

Let m_1, m_2, \dots, m_s be pairwise coprime positive integers, i.e., $\gcd(m_i, m_j) = 1$ for all $i \neq j$. For any given integers a_1, a_2, \dots, a_k and

$$M := \prod_{i=1}^s m_i \quad (8)$$

that simultaneously satisfy the system of congruences:

$$\begin{aligned} x &\equiv a_1 \pmod{m_1}, \\ x &\equiv a_2 \pmod{m_2}, \\ &\vdots \\ x &\equiv a_s \pmod{m_s}. \end{aligned}$$

Then, x is uniquely determined modulo M . The solution can be explicitly constructed as

$$x \equiv \sum_{i=1}^s a_i M_i y_i \pmod{M},$$

where $M_i = M/m_i$ and y_i are the modular multiplicative inverses of M_i modulo m_i , that is, $M_i y_i \equiv 1 \pmod{m_i}$.

We apply this technique to matrix multiplication. Let $A' \in \mathbb{Z}^{p \times q}$ and $B' \in \mathbb{Z}^{q \times r}$. Let

$$C_i \equiv A' B' \pmod{m_i}.$$

Then, we have

$$\begin{aligned} C &\equiv A' B' \pmod{M} \\ &= \sum_{i=1}^s C_i M_i y_i \pmod{M}. \end{aligned} \quad (9)$$

In this paper, we use this approach, a direct reconstruction using Chinese Remainder Theorem. An alternative approach is to use the Garner method.

3 Proposed Method

In this section, we propose a GEMM-based method that combines the Chinese Remainder Theorem and error-free matrix multiplication, which we refer to as *Ozaki scheme II*. Let $A \in \mathbb{F}^{p \times q}$ and $B \in \mathbb{F}^{q \times r}$. We prepare two diagonal matrices D and E whose diagonal elements are powers of two, and we have

$$C = AB = D^{-1} DABEE^{-1} \approx D^{-1} A' B' E^{-1}, \quad (10)$$

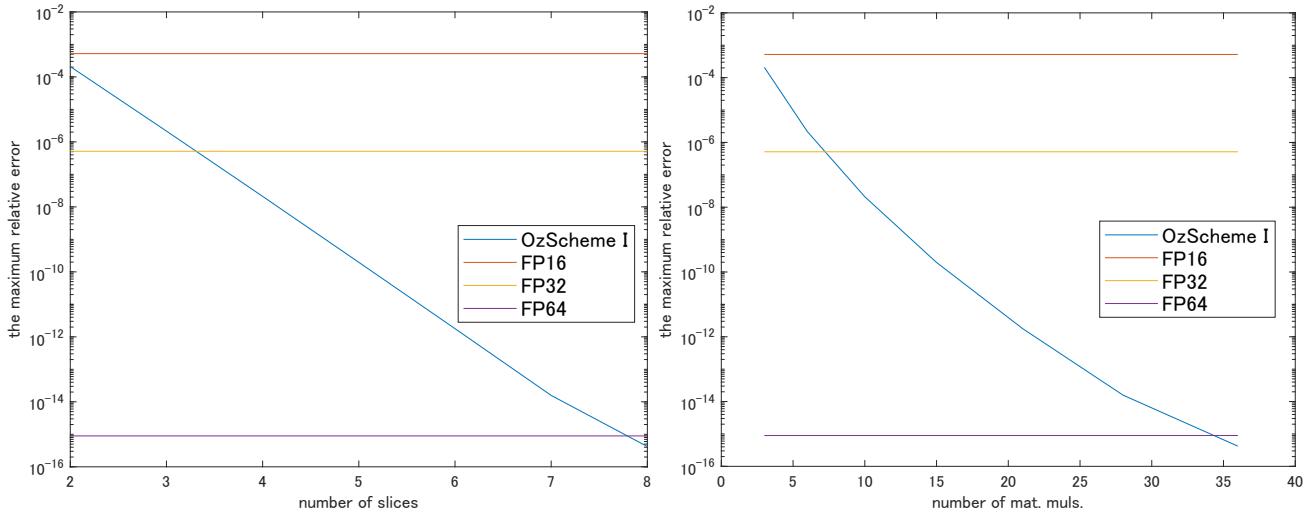


Figure 1. Maximum relative error for each slice (left) and the number of matrix multiplications (right)

where

$$DA \approx A' \in \mathbb{Z}_{k_A}^{p \times q}, \quad BE \approx B' \in \mathbb{Z}_{k_B}^{q \times r}. \quad (11)$$

We first set s , the number of matrix multiplications. We next pick up $m_i \geq 2$ for $1 \leq i \leq s$ from a table, where m_1, m_2, \dots, m_s are coprime to each other, and set M as (8). Matrices $A'_t \in \mathbb{Z}^{p \times q}$ and $B'_t \in \mathbb{Z}^{q \times r}$ are generated as follows:

$$\begin{aligned} (a'_t)_{ij} &\equiv a'_{ij} \pmod{m_t}, \quad -\frac{m_t}{2} \leq (a'_t)_{ij} \leq \frac{m_t}{2}, \\ (b'_t)_{ij} &\equiv b'_{ij} \pmod{m_t}, \quad -\frac{m_t}{2} \leq (b'_t)_{ij} \leq \frac{m_t}{2} \end{aligned} \quad (12)$$

for $1 \leq t \leq s$. If we set proper m_i , then we can compute $A'_i B'_t$ without rounding errors using a BLAS routine. This point is discussed in detail in Sections 3.1 and 3.2, using INT8 TCs and FP64 as examples.

Let $C_i \equiv A'_i B'_i \pmod{m_i}$ and $Y = \sum_{i=1}^s C_i M_i y_i$. Then, we have

$$C' \equiv A' B' \pmod{M} = Y \pmod{M}. \quad (13)$$

Therefore, the candidates of c'_{ij} are

$$\dots, y_{ij} - 2M, y_{ij} - M, y_{ij}, y_{ij} + M, y_{ij} + 2M, \dots$$

Assume that for all (i, j) pairs

$$c_{\min} \leq (A' B')_{ij} \leq c_{\max}.$$

If $c_{\max} - c_{\min} < M$, we find the unique C' in (13). For simplicity, let

$$c_{\max} = -c_{\min} := \max_{i,j} (|A'| |B'|)_{ij}.$$

If

$$2c_{\max} < M \quad (14)$$

is satisfied, we can find the matrix $X \in \mathbb{Z}^{p \times r}$ such that

$$-\frac{M}{2} < -c_{\max} \leq X_{ij} \leq c_{\max} < \frac{M}{2}. \quad (15)$$

If M is smaller than or equal to $2c_{\max}$, we may find multiple candidates of the result (see Fig. 2). Although c_{\max} is defined as the maximum over all elements for simplicity, an element-wise definition is also possible.

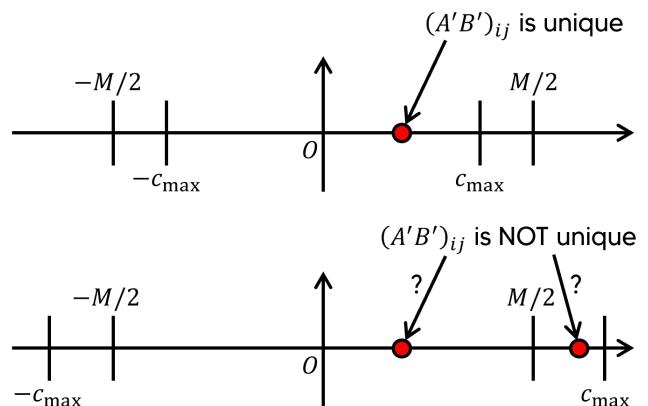


Figure 2. Unique / Non-unique candidate for $A' B'$

The constants k_A and k_B in (11) are important for the accuracy of the computed result because it is strongly related to the truncation error. Here we explain a simple way to find k_A and k_B . From (11),

$$\max_{i,j} (|A'| |B'|)_{ij} \leq q 2^{k_A + k_B} < \frac{M}{2}. \quad (16)$$

From (16), we set $k_A + k_B$ in (11) as

$$k_A + k_B := \left\lfloor \log_2 \frac{M/2 - 1}{q} \right\rfloor. \quad (17)$$

Then, we can set the diagonal matrices D and E in (11). If A and B have the same precision, we assume $k_A = k_B$, leading to the following expression:

$$k_A = k_B := \left\lfloor \frac{1}{2} \log_2 \frac{M/2 - 1}{q} \right\rfloor. \quad (18)$$

If the precisions of the matrices A and B differ, for instance, if A is in FP32 and B is in FP64, we assume $k_A < k_B$. If s increases, M also increases, and as a result, $k_A + k_B$ becomes larger. Note that

- if the matrices are sparse, q in (16) can be reduced,
- $q2^{k_A+k_B}$ in (16) is overestimated as the upper bound of c_{\max} . Alternative ways are to use the Cauchy-Schwarz inequality or to use low-precision computation (see Section 3.1),
- if the range of $A'B'$ is known, that is, c_{\min} and c_{\max} , we can improve (17).

Overall, Ozaki scheme II consists of the following four parts:

Part 1: computation of k_A and k_B from the given s , A , and B . Then obtain A' and B' as in (11).

Part 2: repetition of the following for $i = 1, \dots, s$

Part 2-a: determination of A'_i and B'_i as in (12).

Part 2-b: computation of $C_i := A'_i B'_i$ using a BLAS routine.

Part 2-c: computation of $Z := Z + C_i M_i y_i$.

Part 3: determination of the unique candidate X from the matrix Z .

Part 4: application of the inverse scaling $D^{-1} X E^{-1}$ in (10).

Note that $M_i y_i$ and m_i are calculated in advance and stored in a table. In Part 2-b, we use appropriate functions such as GEMM, TRMM, or SYRK depending on the structure of the matrices A and B . In Part 3, we find x_{ij} in (15) by a range reduction, so high-precision computation is essential in Part 2-c. In Ozaki scheme I, we keep A_1, \dots, A_k and B_k, \dots, B_ℓ , which consume a lot of memory. However, Ozaki scheme II immediately discards the matrices A'_i and B'_i after obtaining C_i .

3.1 Using INT8 TCs for matrix multiplication

If we use INT8 TCs on GPU, we set $2 \leq m_i \leq 256$ for $1 \leq i \leq s$ and m_1, m_2, \dots, m_s to be coprime to each other.

Under these conditions, we prepare the largest possible m_1, m_2, \dots, m_s . For example, when $s = 16$, we set m_i as

$$\begin{aligned} m := & (256, 255, 253, 251, \\ & 247, 239, 233, 229, \\ & 227, 223, 217, 211, \\ & 199, 197, 193, 191)^T \in \mathbb{N}^{16}. \end{aligned} \quad (19)$$

These numbers and $M_i y_i$ in (9) are stored in a table for $s = 2, 3, \dots$. The result of modular arithmetic with any number modulo m_i falls within the range -128 to 127 , which is represented in INT8. In the case where the result of a modulo 256 operation is 128, the wraparound behavior of the INT8 type maps this value to -128 , thereby avoiding any issues. Then no error occurs in $A'_i B'_i$ for $q < 2^{17}$ using cublasGemmEx because the result is stored by INT32. If $q \geq 2^{17}$, it is possible to apply block matrix multiplication for error-free matrix multiplication. The code has been open-sourced by Uchino (2025).

Figure 3 shows $k := k_A = k_B$ in (18) for $p = q = r \in \{1024, 4096, 16384\}$. For $s = 16$, $k = 53$ is expected to be achieved, allowing FP64 emulation with 16 matrix multiplications. Here, this expectation is true if there are no significant differences in the absolute values between the elements of the matrix. To achieve results comparable to double-precision arithmetic, Ozaki scheme I requires 7 to 8 slices, corresponding to 28 to 35 matrix multiplications. This observation suggests the potential advantage of Ozaki scheme II.

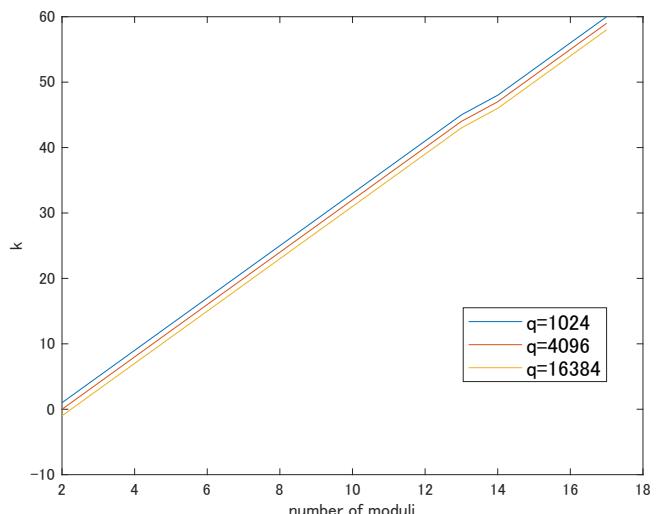


Figure 3. $k := k_A = k_B$ in (18) from number of moduli using INT8 Tensor Cores for matrix multiplication

Algorithm 1 Outline of proposed method using INT8 tensor cores. The functions $\text{trunc}(\cdot)$ and $\text{round}(\cdot)$ round inputs into integers in round-towards-zero and round-to-nearest-even mode, and E is all-ones matrix of appropriate size. \mathbb{F}_{64} is a set of floating-point numbers in FP64.

Input: $A \in \mathbb{F}_{64}^{p \times q}, B \in \mathbb{F}_{64}^{q \times r}, m \in \mathbb{N}^s, s \in \mathbb{N}$

Convert FP64 to INT8:

- 1: Determine shift values $D \in \mathbb{F}_{64}^{p \times q}$ and $E \in \mathbb{F}_{64}^{q \times r}$
- 2: $A' := \text{trunc}(DA)$ $\{A' \in \mathbb{F}_{64}^{p \times q} \cap \mathbb{Z}^{p \times q}\}$
- 3: $B' := \text{trunc}(BE)$ $\{B' \in \mathbb{F}_{64}^{q \times r} \cap \mathbb{Z}^{q \times r}\}$
- 4: $(a'_t)_{ij} := a'_{ij} \bmod m_t$ ($1 \leq t \leq s$)
- 5: $(b'_t)_{ij} := b'_{ij} \bmod m_t$ ($1 \leq t \leq s$)

Matrix multiplication using INT8 TCs:

- 6: $C'_t := A'_t B'_t$ ($1 \leq t \leq s$)

Convert matrix products into UINT8:

- 7: $(c''_t)_{ij} := (c'_t)_{ij} - \lfloor (c'_t)_{ij} / m_t \rfloor m_t$ ($1 \leq t \leq s$)

Accumulate matrix products and inversely scale:

- 8: $C''' := \sum_{t=1}^s C''_t \cdot My_t / m_t$
- 9: $C''' := C''' \bmod M$
- 10: $C := D^{-1} C''' E^{-1}$

Output: $C \in \mathbb{F}_{64}^{p \times r}$

3.2 Using FP64 for matrix multiplication

We set prime numbers m_1, m_2, \dots, m_s as

$$qm_i^2 \leq 4u^{-1}, \quad 1 \leq i \leq s. \quad (20)$$

For $A'_t \in \mathbb{Z}^{p \times q}$ and $B'_t \in \mathbb{Z}^{q \times r}$ in (12), using (20), we have

$$(|A'_t||B'_t|)_{ij} \leq q \cdot \frac{1}{2}m_t \cdot \frac{1}{2}m_t \leq u^{-1}, \quad u = 2^{-53} \quad (21)$$

for all (i, j) pairs and $1 \leq t \leq s$. Hence, no rounding error occurs in $A'_i B'_i$ for $1 \leq i \leq s$ using GEMM in BLAS. Similarly, considering (16), $k_A + k_B$ is obtained as in (17).

Note that m_i for $1 \leq i \leq s$ does not need to be a prime number as long as they are pairwise coprime. However, unlike the case of INT8 TCs, when using FP64, they are simply chosen to be prime numbers to ensure that sufficiently large primes can be easily found. For example, for $s = 16$ and $n = 2^{10}$, we set m_i from (20) as

$$\begin{aligned} m := & (4194301, 4194287, 4194277, 4194271, \\ & 4194247, 4194217, 4194199, 4194191, \\ & 4194187, 4194181, 4194173, 4194167, \\ & 4194143, 4194137, 4194131, 4194107)^T \in \mathbb{N}^{16}. \end{aligned} \quad (22)$$

The reason is that from (20),

$$m_i^2 \leq \frac{2^{55}}{2^{10}},$$

so that

$$m_1 \approx 2^{22} \leq \sqrt{2^{45}}.$$

Note that m in (22) satisfies

$$\frac{m_{16}}{m_1} = \frac{4194107}{4194301} = 0.99995 \dots .$$

It is expected that

$$M \approx sm_s.$$

Because M is proportional to s , $k_A + k_B$ is also proportional to s . In contrast, m in (19) satisfies

$$\frac{m_{16}}{m_1} = \frac{191}{256} = 0.74 \dots .$$

The growth rate of M decreases as s increases.

Here, we deal with the application of FP64 to multi-word arithmetic. Ozaki scheme II can be applied to any multi-word format; we assume that matrices are represented as an unevaluated sum of v floating-point matrices such as

$$A := \sum_{i=1}^v A^{(i)}, \quad B := \sum_{i=1}^v B^{(i)}, \quad (23)$$

where for $1 \leq i \leq v-1$,

$$u|A^{(i)}| \geq |A^{(i+1)}|, \quad u|B^{(i)}| \geq |B^{(i+1)}|. \quad (24)$$

Again, we set diagonal matrices D and E such that

$$C \approx D^{-1} A' B' E^{-1}, \quad A' \approx DA, \quad B' \approx BE.$$

In Part 2-a, we compute

$$\begin{aligned} A'_i &\approx \left(D \sum_{i=1}^v A^{(i)} \right) \bmod m_i, \\ B'_i &\approx \left(\left(\sum_{i=1}^v B^{(i)} \right) E \right) \bmod m_i, \end{aligned}$$

where $A'_i \in \mathbb{Z}^{p \times q}$ and $B'_i \in \mathbb{Z}^{q \times r}$ for all i . The rest of the parts are the same as in the original.

Figure 4 illustrates the behavior of the constant $k := k_A = k_B$ as a function of the number of moduli s . When s is in the range of 16 to 20, the value of k is approximately 160, which corresponds to a precision level equivalent to that obtained by triple-word arithmetic. As s increases to the range of 21 to 25, k increases to approximately 210, indicating a precision level comparable to that of quadruple-word arithmetic.

Thus far, we have considered the product of two matrices; however, the same approach can be extended to the product of three or more matrices. As an example, when dealing with the product of three matrices A , B , and C , we convert them

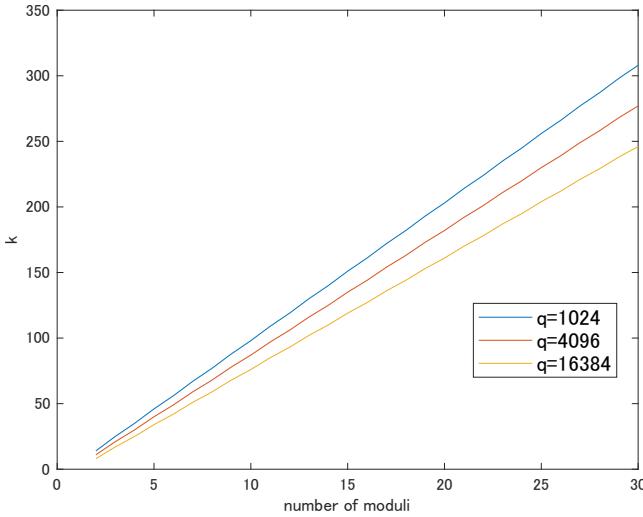


Figure 4. $k := k_A = k_B$ from number of moduli (18) using FP64 for matrix multiplication

to

$$DA \approx A' \in \mathbb{Z}_{k_A}^{p \times q}, \quad BE \approx B' \in \mathbb{Z}_{k_B}^{q \times r}, \\ E^{-1}CF \approx C' \in \mathbb{Z}_{k_C}^{r \times w},$$

where the matrices D , E , and F are diagonal, with diagonal entries that are powers of two, and k_A , k_B , and k_C are determined to satisfy

$$qr2^{k_A+k_B+k_C} < \frac{M}{2}.$$

Then a similar procedure to that in (16) and (17) can be applied. If $k_A = k_B = k_C$, we set

$$k_A = k_B = k_C := \left\lfloor \frac{1}{3} \log_2 \frac{M/2 - 1}{qr} \right\rfloor.$$

4 Numerical Experiment

In this section, we present considerations on both accuracy and computational performance based on the results of numerical experiments. We report experimental results on FP64 emulation using INT8 TCs, as well as on quad-word format emulation using FP64 on a CPU. In both scenarios, we assume that the precision levels are equal, that is, $k_A = k_B$.

4.1 Using INT8 TCs for FP64 emulation

All numerical experiments here were conducted on NVIDIA GH200 Grace Hopper Superchip and NVIDIA GeForce RTX 4090 GPU with NVIDIA CUDA Toolkit 12.8.61. The tested methods will be denoted as follows:

- DGEMM: cublasGemmEx with CUDA.R_64F

- OS II-fast- s : Algorithm 1 with s moduli, employing the Cauchy–Schwarz inequality for the line 1 to satisfy the condition (14) for outputs the unique result
- OS II-accu- s : Algorithm 1 with s moduli, employing cublasGemmEx with CUDA.R_64F for the line 1 to satisfy the condition (14) for outputs the unique result
- ozIMMU_EF-S: Ozaki scheme I (implemented by Uchino (2024)) with S slices

The test matrices $A \in \mathbb{F}^{p \times q}$ and $B \in \mathbb{F}^{q \times r}$ were generated as

$$a_{ij}, b_{ij} \approx (\text{rand} - 0.5) \cdot \exp(\phi \cdot \text{randn}),$$

where $\phi \in \mathbb{R}$ controls the exponent distribution, $\text{rand} \in (0, 1] \subset \mathbb{R}$ is a uniformly distributed random number and $\text{randn} \in \mathbb{R}$ is drawn from the standard normal distribution. The non-negative constant ϕ specifies the tendency to difficulty in terms of the accuracy of matrix multiplication. Empirically, the exponent distribution for matrix multiplication in HPL is comparable to $\phi = 0.5$.

Figure 5 shows the accuracy of DGEMM, OS II-fast- s , and OS II-accu- s for $s \in \{8, 9, \dots, 20\}$ on GH200. We obtained similar results on RTX 4090. For reference, ozIMMU_EF-S requires $S \geq 8$ to obtain comparable or slightly more accurate results than DGEMM (cf. numerical results by Ootomo et al. (2024) or Uchino et al. (2025)). The initial estimation at the line 1 of Algorithm 1 strongly affects the truncation error at the lines 2 and 3. Thus, OS II-accu returns more accurate results than OS II-fast due to less overestimation of the upper bound of $|A'||B'|$ in (14) by direct matrix multiplication using INT8 TCs. For $\phi = 0.5$, both proposed methods require 14 or 15 moduli to achieve the accuracy of the DGEMM level. OS II-accu can deal with larger ϕ to produce sufficiently accurate results.

Tables 3 and 4 show the throughput performance of DGEMM, ozIMMU_EF-8, OS II-fast- s , and OS II-accu- s for $s \in \{14, 15, 16, 17, 18\}$. For small problems, matrix multiplication using INT8 TCs did not achieve sufficient performance. Additionally, the overhead of operations except for matrix multiplication became relatively significant, resulting in substantially lower performance compared to DGEMM. However, as shown in Table 3, our implementation achieved 80.2 TFLOPS for $p = q = r = 16384$ and $s = 14$ on GH200. These results demonstrate that the proposed methods achieved higher throughput than DGEMM while attaining DGEMM-level accuracy.

Figures 6 and 7 show the time breakdown of OS II-fast- s and OS II-accu- s for $s = 2, 3, \dots, 20$. For small problems, the overhead of kernel launches becomes a performance

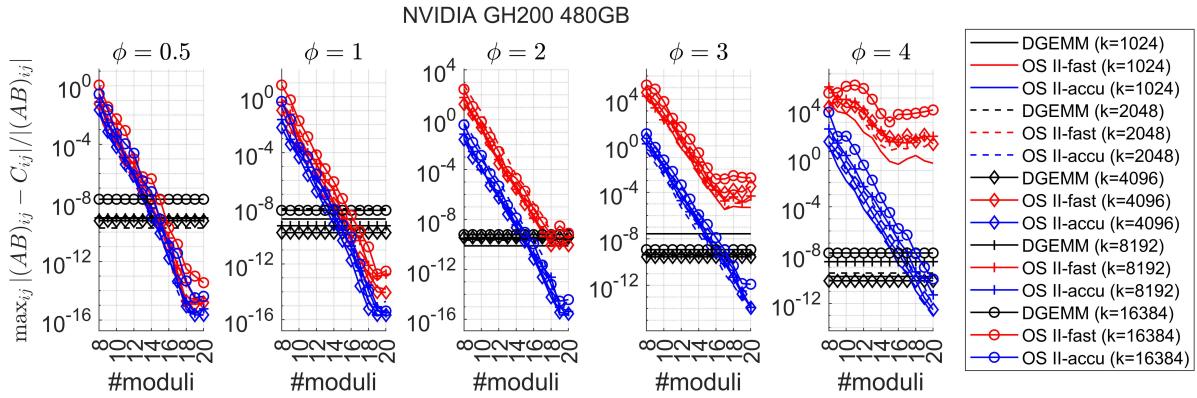


Figure 5. Accuracy comparison for $p = r = 1024$ and $q \in \{1024, 2048, 4096, 8192, 16384\}$ on NVIDIA GH200 Grace Hopper Superchip

Table 3. Throughput in TFLOPS on NVIDIA GH200 Grace Hopper Superchip

Methods \ Matrix size	2048	4096	8192	16384
DGEMM (cuBLAS)	56.9	61.3	62.0	60.9
OS II-fast-14	16.5	48.6	72.1	80.2
OS II-fast-15	15.4	45.2	67.4	74.6
OS II-fast-16	14.6	43.2	63.8	70.4
OS II-fast-17	13.8	40.5	60.1	66.1
OS II-fast-18	13.0	38.3	56.9	62.6
OS II-accu-14	14.7	42.7	64.7	71.1
OS II-accu-15	14.0	40.2	60.9	66.9
OS II-accu-16	13.3	38.5	58.0	63.3
OS II-accu-17	12.6	36.4	55.1	59.9
OS II-accu-18	12.0	34.7	52.2	56.6
ozIMMU_EF-8	12.4	25.6	29.9	34.5

Table 4. Throughput in TFLOPS on NVIDIA RTX 4090

Methods \ Matrix size	2048	4096	8192
DGEMM (cuBLAS)	0.61	0.62	0.62
OS II-fast-14	3.73	6.94	9.81
OS II-fast-15	3.54	6.55	9.20
OS II-fast-16	3.31	6.22	8.76
OS II-fast-17	3.07	5.90	8.29
OS II-fast-18	2.91	5.56	7.83
OS II-accu-14	3.46	6.51	9.23
OS II-accu-15	3.29	6.04	8.68
OS II-accu-16	3.10	5.69	8.26
OS II-accu-17	2.93	5.40	7.81
OS II-accu-18	2.84	5.18	7.41
ozIMMU_EF-8	4.22	4.76	5.84

bottleneck. For medium-sized problems, while the kernel launch overhead is mitigated, the conversion from double-precision matrices to INT8 matrices remains the main bottleneck, and the adverse effect of the accumulation of matrix products (conv_32i_2.8u and inverse_scaling) is also not negligible. For sufficiently large problems, both the kernel launch overhead and the accumulation of matrix products become nearly negligible; however, the conversion from double-precision matrices to INT8 matrices continues to be the dominant bottleneck.

The peak performance of INT8 TCs on the GH200 is 1979 TOPS, while the peak performance of FP64 TCs is 67 TFLOPS. On GPUs such as the B200, the peak performance of INT8 TCs increases to 4500 TOPS, whereas the FP64 TCs achieve 40 TFLOPS. In such environments, where INT8 TCs performance improves while FP64 TCs performance decreases compared to the GH200, emulation techniques are expected to become increasingly important.

4.2 Using FP64 for Quad-word arithmetic emulation

The computational environment consisted of a PC with an Intel® Core™ i7-8665U processor (4 cores) and an Intel® Core™ i9-10980XE processor (18 cores), MATLAB 2024b, and Windows 10. Hereafter, we simply refer to them as the Core i7 and Core i9, respectively. The code was implemented as a MATLAB executable (MEX) and compiled with Visual Studio 2022. The compiler flags `/openmp` and `/arch:AVX2` were used for the Core i7, and `/openmp` and `/arch:AVX512` were used for the Core i9.

We set $v = 4$ in (23). It indicates that

$$\begin{aligned} A &= A^{(1)} + A^{(2)} + A^{(3)} + A^{(4)}, \\ B &= B^{(1)} + B^{(2)} + B^{(3)} + B^{(4)}. \end{aligned}$$

$A^{(1)}$ and $B^{(1)}$ are generated by the `randn` function on MATLAB, and (24) are satisfied. We use six-word arithmetic for Part 2-c and Part 3 in Ozaki scheme II. Figure 8 shows the maximum relative errors for Ozaki schemes I and II for $p = q = r \in \{1000, 8000, 15000\}$ on the Core-i9. When the number of matrix multiplications is small, the accuracy of Ozaki scheme II is worse than that of Ozaki scheme I. However, when the number of matrix multiplications increases, Ozaki scheme II shows better accuracy than Ozaki scheme I. This is because the accuracy

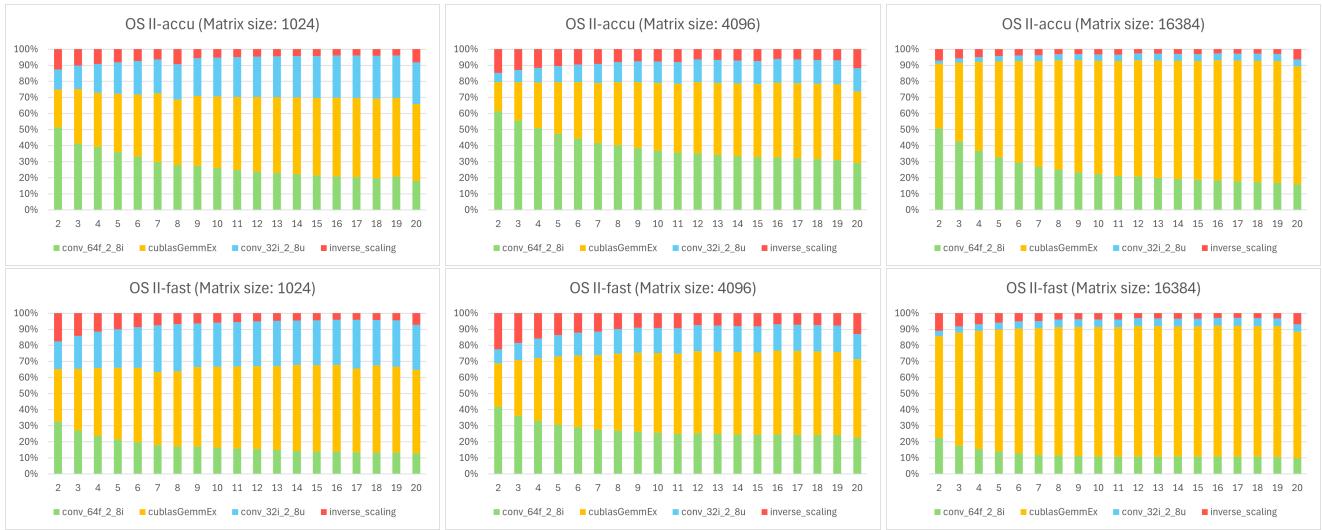


Figure 6. Time breakdown for $p = r = q \in \{1024, 4096, 16384\}$ on NVIDIA GH200 Grace Hopper Superchip. Horizontal values represent number of moduli.



Figure 7. Time breakdown for $p = r = q \in \{2048, 4096, 8192\}$ on NVIDIA GeForce RTX 4090 GPU. Horizontal values represent number of moduli.

of Ozaki scheme II improves linearly with respect to the number of matrix multiplications on a logarithmic scale. The reason is that $\max_i m_i \approx \min_i m_i$ is satisfied, as explained in the previous section.

Figures 9 and 10 show the proportion of computation time for each part across different matrix sizes. Although the computation time of Part 2-b is ideally expected to be dominant, it is evident that it does not account for the largest portion of the total computation time when $n = 1000$. As the matrix size increases, the proportion of time spent on matrix multiplication becomes higher, indicating that the method becomes more dependent on GEMM.

Tables 5 and 6 show the throughput in GFLOPS when using the Core i7 and Core i9, respectively. Ozaki scheme I-10 and Ozaki scheme II-22 exhibit comparable accuracy. However, as shown in Table 5, Ozaki scheme II is

slower when $n = 1000$, whereas it demonstrates superior performance for values of n other than 1000. Ozaki scheme II achieved a speedup of up to 2.29x on the Core i7 at $n = 4000$ and up to 2.12x on the Core i9 at $n = 8000$.

Table 5. Throughput in GFLOPS for $p = q = r \in \{1000, 4000, 8000\}$ on Intel® Core™ i7-8665U processor

Methods \ Matrix size	1000	4000	8000
DGEMM	99.5	105	114
OS I-6	2.32	4.35	4.32
OS I-7	2.32	3.33	3.02
OS I-8	2.32	2.63	2.31
OS I-9	1.96	2.11	1.88
OS I-10	1.60	1.75	1.51
OS II-19	1.35	3.45	4.10
OS II-20	1.31	2.99	3.58
OS II-21	1.18	2.89	3.45
OS II-22	1.20	2.81	3.47

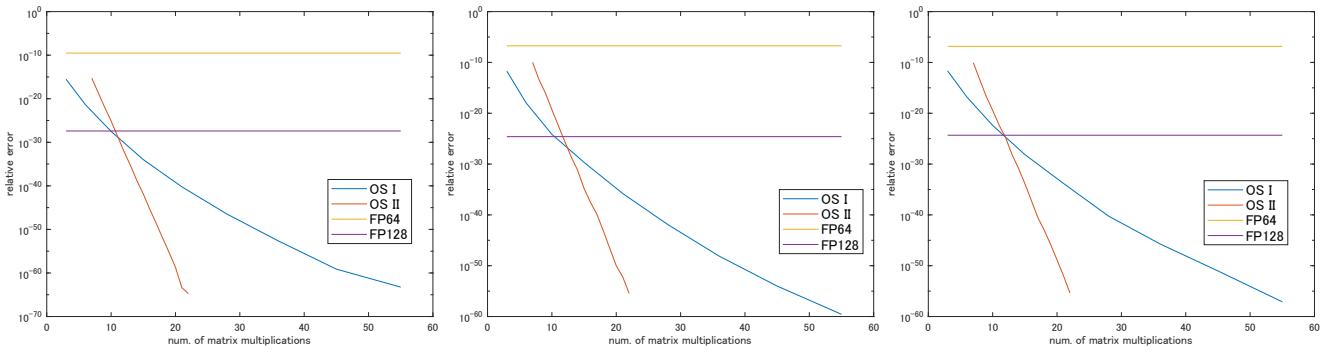


Figure 8. Accuracy comparison for $p = r = q \in \{1000, 8000, 15000\}$ on Intel® Core™ i9-10980XE processor. Horizontal values represent number of moduli.

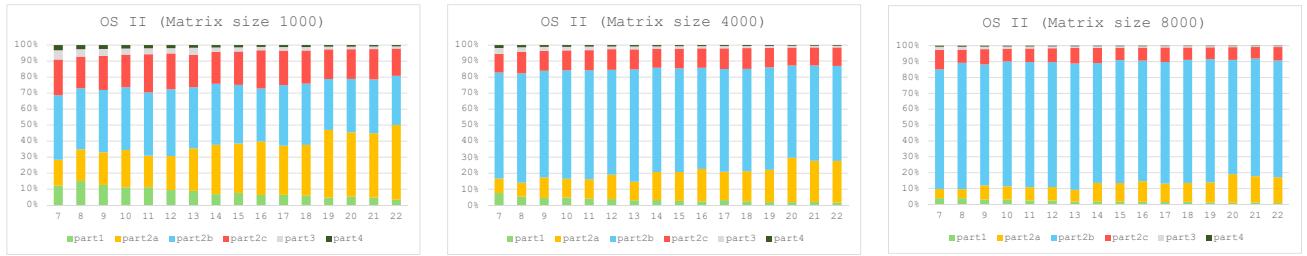


Figure 9. Time breakdown for $p = r = q \in \{1000, 4000, 8000\}$ on Intel® Core™ i7-8665U processor. Horizontal values represent number of moduli.

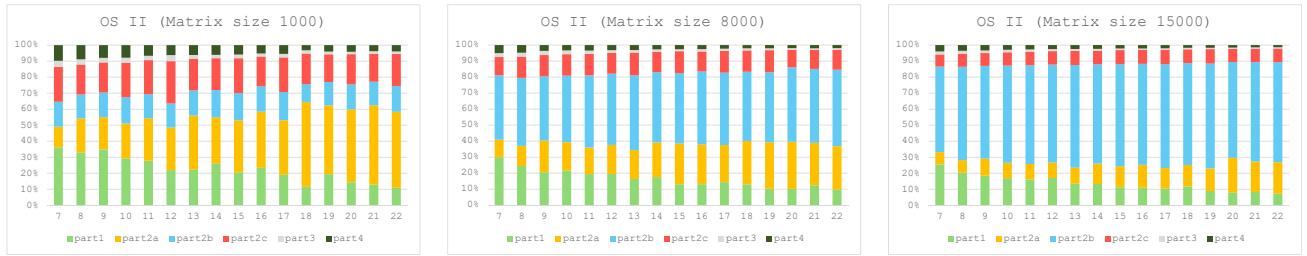


Figure 10. Time breakdown for $p = r = q \in \{1000, 8000, 15000\}$ on Intel® Core™ i9-10980XE processor. Horizontal values represent number of moduli.

Table 6. Throughput in GFLOPS for $p = q = r \in \{1000, 8000, 15000\}$ on Intel® Core™ i9-10980XE processor

Methods \ Matrix size	1000	8000	15000
DGEMM	228	1073	1092
OS I-6	6.51	29.6	30.5
OS I-7	5.42	23.5	25.1
OS I-8	4.59	20.3	21.0
OS I-9	4.24	17.0	17.9
OS I-10	3.79	12.4	15.1
OS II-19	4.46	27.3	33.7
OS II-20	4.54	26.2	30.2
OS II-21	4.19	26.6	29.8
OS II-22	4.31	26.3	28.8

5 Conclusion

In this study, we developed a novel algorithm, referred to as Ozaki scheme II, for emulating matrix multiplication. Furthermore, the effectiveness of multi-component arithmetic was also demonstrated. Future challenges include further optimization of the implementation and a detailed analysis of the rounding errors introduced by the proposed method, which will be addressed in future work. We also plan to conduct further experiments on GPUs with higher INT8 Tensor Core performance, such as the B200, to explore the potential of our method on next-generation architectures.

Declaration of conflicting interests

The authors declare no conflict of interest.

Funding

This study was partially supported by the JSPS Grant-in-Aid for Research Activity Start-up No. 24K23874, and the JSPS KAKENHI Grant Nos. 23K28100, 25H01109, and 25K03126.

Supplemental material

Not applicable.

References

- Bailey DH, Hida Y, Li XS and Thompson B (2002) Qd library: Quad-double and double-double arithmetic. <https://www.davidhbailey.com/dhbsoftware/>. Version 2.3.22, accessed January 2025.
- Dawson W, Ozaki K, Domke J and Nakajima T (2024) Reducing numerical precision requirements in quantum chemistry calculations URL <https://arxiv.org/abs/2407.13299>.
- Dongarra J, Gunnels J, Bayraktar H, Haidar A and Ernst D (2024) Hardware trends impacting floating-point computations in scientific applications URL <https://arxiv.org/abs/2411.12090>.
- Fousse L, Hanrot G, Lefèvre V, Péllissier P and Zimmermann P (2025) Mpfr: A multiple-precision binary floating-point library with correct rounding. <https://www.mpfr.org/>. Version 4.2.1, accessed January 2025.
- Hida Y, Li XS and Bailey DH (2001) Algorithms for quad-double precision floating point arithmetic. In: *Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001*. IEEE, pp. 155–162.
- Higham N (2018) A multiprecision world. *SIAM News* 51(4). URL <https://sineWS.siam.org/Details-Page-a-multiprecision-world>.
- IEEE Computer Society (2019) IEEE standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)* DOI:10.1109/IEEEESTD.2019.8766229.
- Mukunoki D, Ozaki K, Ogita T and Imamura T (2020) Dgemm using tensor cores, and its accurate and reproducible versions. In: Sadayappan P, Chamberlain BL, Juckeland G and Ltaief H (eds.) *High Performance Computing*. Cham: Springer International Publishing, pp. 230–248.
- Muller JM, Revol N and Langlois P (2015) Efficient floating-point arithmetic using triple-word numbers. *Mathematics of Computation* 84(292): 1419–1437. DOI:10.1090/S0025-5718-2014-02879-0.
- Nakata M (2022) *MPLAPACK version 2.0.1 user manual*. URL <https://arxiv.org/abs/2109.13406>.
- NVIDIA Corporation (2024) NVIDIA Tensor Cores. URL <https://www.nvidia.com/en-us/data-center/tensor-cores/>.
- Ootomo H (2024) ozIMMU - DGEMM on Int8 Tensor Core. URL <https://github.com/enpls0/ozIMMU>.
- Ootomo H, Ozaki K and Yokota R (2024) Dgemm on integer matrix multiplication unit. *The International Journal of High Performance Computing Applications* in Press. DOI:10.1177/10943420241239588.
- Ozaki K, Mukunoki D and Ogita T (2025) Extension of accurate numerical algorithms for matrix multiplication based on error-free transformation. *Japan Journal of Industrial and Applied Mathematics* 42: 1–20. DOI:10.1007/s13160-024-00677-z.
- Ozaki K, Ogita T, Oishi S and Rump SM (2012) Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. *Numerical Algorithms* 59(1): 95–118.
- Ozaki K, Ogita T, Oishi S and Rump SM (2013) Generalization of error-free transformation for matrix multiplication and its application. *Nonlinear Theory and Its Applications, IEICE* 4(1): 2–11.
- Uchino Y (2024) Accelerator for ozIMMU. R-CCS github repository. URL https://github.com/RIKEN-RCCS/accelerator_for_ozIMMU.
- Uchino Y (2025) GEMMul8: GEMM emulation using int8 matrix engines based on the Ozaki Scheme II. R-CCS github repository. URL <https://github.com/RIKEN-RCCS/GEMMul8>.
- Uchino Y, Ozaki K and Imamura T (2025) Performance enhancement of the ozaki scheme on integer matrix multiplication unit. *The International Journal of High Performance Computing Applications* DOI:10.1177/10943420241313064. URL <https://doi.org/10.1177/10943420241313064>.

Author Biographies

Katsuhisa Ozaki is a full professor in the Department of Mathematical Sciences at the Shibaura Institute of Technology. He received his Ph.D. in engineering from Waseda University in 2007. He was an Assistant Professor (2007-2008) and a Visiting Lecturer (2008-2009) at Waseda University. At Shibaura Institute of Technology, he has served as Assistant Professor (2010-2013) and Associate Professor (2013-2019) and currently is a Professor since 2019. His research interests include reliable computing, particularly addressing rounding error problems in finite-precision arithmetic. He mainly focuses on numerical linear algebra and develops fast and accurate algorithms.

Yuki Uchino is a postdoctoral researcher at RIKEN R-CCS. He received his Ph.D. in engineering from Shibaura Institute of Technology in 2024. His research interests include

reliable computing, numerical linear algebra, and highly accurate algorithms.

Toshiyuki Imamura is a team principal of the Large-scale Parallel Numerical Computing Technology Team at RIKEN R-CCS, and is responsible for developing numerical libraries on Fugaku. He received his Diploma and Doctorate in Applied Systems and Sciences from Kyoto University in 1993 and 2000. He was a Researcher at CCSE, JAERI (1996-2003), a visiting scientist at HLRS (2002), and an associate professor at the University of Electro-Communications (2003-2012). His research interests include HPC, auto-tuning technology, and parallel eigenvalue computation. His research group won the HPL-MxP ranking (2020-2021) and was nominated as the Gordon Bell Prize finalist in SC05, SC06, and SC20.