The 2M Multiplication Algorithm for Complex Matrices

Peter Caday
NVIDIA Corporation
pcaday@nvidia.com

June 9, 2026

Abstract

Complex matrix multiplication is typically computed using 4 real matrix multiplications (GEMMs) of the same size. The well-known 3M multiplication algorithm reduces this cost to 3 real GEMMs, together with quadratic time pre- and post-processing steps.

In this paper, we reduce 3M to 2M for matrices with integer real and imaginary parts, performing complex GEMM with only 2 real GEMMs of the same size, along with quadratic time pre- and post-processing. For floating-point matrices, 2M multiplication combines naturally with the Ozaki-II scheme, yielding a practical, high-performance algorithm for computing a complex floating-point GEMM in roughly twice the time of a real GEMM of the same size. As corollaries, we derive new algorithms for symmetric rank-k updates (`SYRK`/`HERK`) that internally use full rectangular GEMMs.

1 Introduction

The classical approach to multiply complex matrices $A = A_r + iA_i \in \mathbb{C}^{m\times k}$ and $B = B_r + iB_i \in \mathbb{C}^{k\times n}$ is to expand the product into 4 real matrix multiplications:

$$C = AB = (A_rB_r - A_iB_i) + i(A_rB_i + A_iB_r).$$

Let us call this the *4M* algorithm, following [7], since it involves 4 real matrix multiplications. 4M multiplication typically forms the backbone of complex floating-point GEMMs in high-performance libraries.

The well-known *3M* algorithm reduces this cost to 3 real matrix multiplications, by first computing $A_r + A_i$ and $B_r + B_i$:

$$C = \big(A_rB_r - A_iB_i\big) + i\big((A_r + A_i)(B_r + B_i) - A_rB_r - A_iB_i\big). \tag{1}$$

Note that while (1) is mathematically exact, in floating-point arithmetic the additions $A_r + A_i$, $B_r + B_i$ can reduce numerical stability [2]. Hence, numerical libraries typically offer 3M GEMMs as an opt-in feature.

For scalar complex products, it can be proven that 3 real multiplications is indeed minimal [8]. In this paper, we will show that we can do better for matrix products.

2 2M Complex Multiplication

To start, we consider input matrices $A = A_r+iA_i$, $B = B_r+iB_i$ with integral real and imaginary parts, and write $AB = C = C_r + iC_i$. Suppose m is an integer chosen large enough that $|\mathrm{Re}(c_{ij})|, |\mathrm{Im}(c_{ij})| \leq \frac{m}{2}$ for each entry c_{ij} of C. Then, if we know C mod m, we can evidently recover C.

Now suppose that m is odd and there is an integer s such that $s^2 = -1 \pmod m$, in effect a "real surrogate" for i. It can be shown that such an s exists iff m is a product of primes congruent to 1 mod 4. We call such an m a *2M modulus*.

Perform the 3M-like preprocessing step

$$A_\pm = A_r \pm sA_i \pmod m, \qquad B_\pm = B_r \pm sB_i \pmod m. \tag{2a}$$

Next, perform the 2 real multiplications

$$C_\pm = A_\pm B_\pm = (A_r B_r - A_i B_i) \pm s(A_r B_i + A_i B_r) \pmod m. \tag{2b}$$

Then it is not hard to see that

$$C = 2^{-1}\big(C_- + C_+\big) + is \cdot 2^{-1}\big(C_- - C_+\big) \pmod m. \tag{2c}$$

Here $2^{-1} = (m+1)/2$ is the inverse of 2 modulo m, explaining why m must be odd. Together, equations (2a–2c) form the *2M multiplication algorithm*. As with 3M, the new algorithm requires $\Theta(mn + mk + nk)$ scalar operations in addition to the matrix multiplications.

2.1 Algebraic Underpinnings

To understand where 2M multiplication comes from, we borrow some basic ideas from algebraic number theory. Algebraically, A, B, C are matrices over the ring of Gaussian integers $\mathbb{Z}[i] = \{x + yi \,|\, x, y \in \mathbb{Z}\}$. Gaussian integers have a rich algebraic structure. In particular, they form a Euclidean domain, meaning that we have well-defined notion of division with remainder; modular arithmetic and the Chinese remainder theorem (CRT) also extend to $\mathbb{Z}[i]$.

Suppose now that $w = u + vi \in \mathbb{Z}[i]$ with $\gcd(u, v) = 1$, and let $m = |w|^2 = u^2 + v^2$. Factoring $m = w\overline{w}$, the CRT allows us to reconstruct $C \bmod m$ from $C \bmod w$, $C \bmod \overline{w}$. We can write

$$C \bmod w = (A \bmod w)(B \bmod w), \qquad C \bmod \overline{w} = (A \bmod \overline{w})(B \bmod \overline{w}).$$

The crucial step is to convert these complex multiplications to real equivalents. Algebraically speaking, the mapping $i \mapsto s = -uv^{-1}$ induces an isomorphism of quotient rings $\mathbb{Z}[i]/w\mathbb{Z}[i] \cong \mathbb{Z}/m\mathbb{Z}$.[1] In other words, working in $\mathbb{Z}[i]$ modulo w is equivalent to working in $\mathbb{Z}$ modulo m.

With this isomorphism, the real matrices $A_\pm$ in (2a) are precisely the residues of A modulo w, $\overline{w}$: $A_+ = A \bmod w$, $A_- = A \bmod \overline{w}$, and similarly for B_+, B_-. Equation (2b) multiplies these residues modulo w, $\overline{w}$, and finally (2c) is the Chinese remainder theorem for recovering $C \bmod m$ from $C_+ = C \bmod w$, $C_- = C \bmod \overline{w}$, under the same isomorphism.

3 2M Multiplication In Floating Point

3.1 The Ozaki-II Scheme

To apply 2M to non-integer GEMMs, we employ the Ozaki-II scheme [5], a family of algorithms for computing floating-point matrix multiplication (GEMM) to arbitrary accuracy. Ozaki-II works by quantizing its inputs to integers, then performing fast integer matrix multiplication with modular arithmetic. By leveraging the dedicated narrow-precision matrix multipliation units available on

[1] Indeed, we have a trivial homomorphism $\mathbb{Z}/m\mathbb{Z} \hookrightarrow \mathbb{Z}[i]/m\mathbb{Z}[i] \to \mathbb{Z}[i]/w\mathbb{Z}[i]$, well-defined since $w \mid m$, and an inverse map induced from $i \mapsto s \bmod w$ by linearity, also well-defined since $s^2 \equiv 1 \bmod m$.

modern GPUs, Ozaki-II can often significantly outperform standard floating-point GEMM while maintaining the same level of accuracy. Moreover, the level of accuracy may be tuned: either reduced, for additional performance; or increased, to provide additional accuracy beyond that available with a standard floating-point implementation.

We briefly restate the method here. Given real matrices $\mathbf{A} = (\mathbf{a}_{ik}) \in \mathbb{R}^{m\times k}$, $\mathbf{B} = (\mathbf{b}_{kj}) \in \mathbb{R}^{k\times n}$, we choose per-row and per-column scaling factors $\sigma \in \mathbb{R}^m$, $\tau \in \mathbb{R}^n$ for $\mathbf{A}$ and $\mathbf{B}$, respectively, and quantize to integer matrices

$$A = \left(\operatorname{round}_{\mathbb{Z}}(\mathbf{a}_{ik}\sigma_i^{-1}\right) \in \mathbb{Z}^{m\times k}, \qquad B = \left(\operatorname{round}_{\mathbb{Z}}(\mathbf{b}_{kj}\tau_j^{-1})\right) \in \mathbb{Z}^{k\times n}.$$

Here, $\operatorname{round}_S(\cdot)$ represents rounding to a nearest element of a set S. Next, we choose a collection of coprime moduli $m_1, \ldots, m_N \in \mathbb{Z}_+$ whose product M is large enough that each element of the integer product $C = AB$ has absolute value at most $M/2$. Then $C = (c_{ij})$ can be uniquely recovered from its residue modulo M, and we do so by computing the products

$$C_\ell = (A \bmod m_\ell)\,(B \bmod m_\ell)\,, \qquad \ell = 1, \ldots, N, \tag{3}$$

and applying the Chinese remainder theorem (CRT) elementwise:

$$C = \sum_{\ell=1}^{N} C_\ell \cdot \left(\frac{M}{m_\ell} \bmod m_\ell\right)^{-1} \frac{M}{m_\ell}.$$

Then $\mathbf{C} = \mathbf{AB} \approx (\tilde{c}_{ij}\sigma_i\tau_j)$. The accuracy of the approximation is determined primarily by M, with relative elementwise error $(c_{ij} - \tilde{c}_{ij}\sigma_i\tau_j)/c_{ij}$ roughly $O(M^{-1/2})$. Achieving full accuracy also requires a proper choice of scaling factors σ, τ; for more detail, see [3, 6].

Ozaki-II derives its speed from computing the products (3) using dedicated narrow-precision matrix multiplication hardware, which guides the choice of the moduli m_i (e.g. $m_i \le 2^8$ for 8-bit integer GEMMs).

3.2 Complex Ozaki-II With 2M

Ozaki-II extends readily to complex-valued inputs $\mathbf{A} = \mathbf{A}_r + i\mathbf{A}_i \in \mathbb{C}^{m\times k}$, $\mathbf{B} = \mathbf{B}_r + i\mathbf{B}_i \in \mathbb{C}^{k\times n}$. In Uchino et al.'s algorithm [4], they first choose (real) scaling factors σ, τ and quantize $\mathbf{A}_r$, $\mathbf{A}_i$, $\mathbf{B}_r$, $\mathbf{B}_i$ to integer matrices as above. The resulting integer matrices are reduced modulo each m_ℓ, multiplied, and then the real and imaginary parts of C are reconstructed separately using CRT. Scaling factors may be chosen using straightforward extensions of any of the existing real algorithms [3, 5].

For better performance, Uchino et al. apply 3M to each small integer multiplication, after modular reduction; since 3M is an algebraic identity, it is valid modulo any m_ℓ.[2] By substituting 2M multiplication for 3M, we immediately derive a 2M Ozaki-II algorithm for complex floating-point GEMMs (Algorithm 1).

Note also that we can readily hybridize the 2M and 3M Ozaki-II methods, using 2M multiplication for moduli m_ℓ that satisfy the 2M conditions (odd, with a square root of -1), and 3M multiplication for the remaining moduli. For a practical implementation, such a hybrid approach allows reaching higher precision in cases where insufficient 2M moduli are available.

[2] It is also possible to apply 3M *prior* to modular reduction, but this has the disadvantage of requiring high-precision integer arithmetic in 3M's pre- and post-proessing steps.

Algorithm 1 – 2M Ozaki-II

1: Fix a set of coprime 2M moduli $m_1, \ldots, m_N$; let $M = \prod m_i$.
2: Choose scaling factors $\sigma \in \mathbb{R}^m$, $\tau \in \mathbb{R}^n$.
3: Quantize input matrices: $A = \left(\text{round}_{\mathbb{Z}[i]}(\sigma_i^{-1}\mathbf{a}_{ik})\right)$, $B = \left(\text{round}_{\mathbb{Z}[i]}(\tau_j^{-1}\mathbf{b}_{kj})\right)$.
4: For each $i = 1, \ldots, N$, reduce A, B modulo m_i and multiply with (2a–2c).
5: Determine C mod M via CRT.
6: Lift C to $\mathbb{Z}[i]^{m\times n}$ by choosing representatives with real and imaginary parts in $[-M/2, M/2)$.
7: Reconstruct $\mathbf{C} \approx (c_{ij}\sigma_i\tau_j) \in \mathbb{C}^{m\times n}$.

3.3 Other 2M Ozaki-II Variants

Algebraically, we may interpret Uchino et al.'s algorithm as the usual Ozaki-II scheme operating over the Gaussian integers, with *real* integer moduli m_i. 2M Ozaki-II works similarly, but with conjugate pairs of *complex* integer moduli.

We note in passing that 2M multiplication and Algorithm 1 have analogues where we replace $\mathbb{Z}[i]$ by another imaginary quadratic number ring. A natural choice is the Eisenstein integers $\mathbb{Z}[e^{2\pi i/3}] = \{a + be^{2\pi i/3} \mid a, b \in \mathbb{Z}\}$, which form a hexagonal grid in the complex plane. By varying the number ring, we vary the set of valid 2M moduli. Asymptotically speaking, half of all primes are 2M moduli w.r.t. any given imaginary quadratic number ring by quadratic reciprocity and Dirichlet's theorem. In practice, however, since moduli are generally bounded by the input range of hardware matrix multiplication units, specific rings may perform slightly better than others, as we will see in the next section.

4 ZGEMM with 8-bit Integer GEMMs

As a concrete example of Algorithm 1, let us consider complex double-precision GEMM (ZGEMM) with 8-bit integer GEMMs underneath. We enumerate a set of coprime 2M moduli in the range $[1, 256]$, chosen using a greedy algorithm:

$$\mathcal{M}_{\text{2M}} = \{241, 233, 229, 221, 205, 197, 193, 181, 173, 157, 149, 137, 113, 109, \ldots, 37, 29\}.$$

Recall that the accuracy of an Ozaki-II method is determined primarily by M, the product of the chosen moduli. The first $N = 16$ moduli of the above sequence, with product $M \approx 2^{117}$, is sufficient in many cases to achieve accuracy comparable to a standard ZGEMM implementation in double-precision arithmetic.

Note that by comparison Ozaki-II real GEMMs, which are not restricted to 2M moduli, have access to a larger set of moduli

$$\mathcal{M}_{\text{real}} = \{256, 255, 253, 251, 247, 241, 239, 233, 229, 227, \ldots\}.$$

Because these moduli decrease less rapidly than 2M moduli, we require only 15 moduli to achieve the same level of accuracy $M \approx 2^{117}$.

The relative sparsity of $\mathcal{M}_{2M}$ w.r.t. $\mathcal{M}_{\text{real}}$ reflects the fact that asymptotically only half of the prime numbers are valid 2M moduli, as observed in Section 3.3. Furthermore, the product of *all* moduli in $\mathcal{M}_{\text{2M}}$ is $M \approx 2^{152}$, much lower than for real Ozaki-II ($M \approx 2^{342}$). If additional precision is required, e.g. due to a large exponent span in the input matrices [3], we may potentially exhaust available 2M moduli. In this case, additional non-2M moduli may be added to the set of moduli using the hybrid 2M/3M algorithm described in Section 3.2.

As mentioned previously, the set of 2M moduli may be varied by discretizing to different imaginary quadratic number rings. Indeed, for 8-bit integer moduli, the Eisenstein integers have a slight advantage over Gaussian integers: we can achieve $M \approx 2^{154.5}$ with 2M moduli. However, this advantage is offset by slightly more complicated quantization/dequantization steps; in particular, quantization involves rounding each point in the complex plane to the nearest point on a hexagonal lattice.

5 2M Rank-k Updates[3]

The 2M method also yields computationally efficient methods for the real and complex BLAS-3 rank-k and rank-$2k$ update routines (`SYRK`, `HERK`, `SYR2K`, `HER2K`). We recall these routines' definitions, omitting alpha and beta scaling for simplicity:

- `SYRK`: $C = AA^T,\ A \in \mathbb{R}^{n\times k}$.
- `HERK`: $C = AA^H,\ A \in \mathbb{C}^{n\times k}$.
- `SYR2K`: $C = AB^T + BA^T,\ A, B \in \mathbb{R}^{n\times k}$.
- `HER2K`: $C = AB^H + BA^H,\ A, B \in \mathbb{C}^{n\times k}$.

In each case C is (Hermitian) symmetric, so only the lower or upper triangle of C is computed and stored. In theory, the symmetry of C reduces the operation count in half (e.g. $n^2k + o(n^2k)$ flops for SYRK vs. $2n^2k$ for a same-sized GEMM), but library implementations generally cannot obtain the full 2× speedup. The reason is that SYRK-like routines are typically implemented with a modified GEMM kernel that masks its output to the lower or upper triangle of the output matrix C [1]. Each kernel operates on a fixed-size tile of C, leading to wasted computation near the diagonal, where tiles lie partly outside the desired triangle.

With 2M, it is possible to implement symmetric products using full, unmasked GEMMs, while still maintaining the expected 2x speedup. The key idea is that Hermitian symmetry in the problem induces a symmetry relation in the real integer products $C_\pm$, allowing us to compute only half as many integer GEMMs. We outline the methods below, which may be applied in either the integer domain (modulo m) or in floating point, in conjuction with Ozaki-II.

`HERK`: Writing $A = A_r + iA_i$ and applying 2M, we have

$$C_+ = (A_r + sA_i)(A_r - sA_i)^T \bmod m,$$
$$C_- = (A_r - sA_i)(A_r + sA_i)^T \bmod m.$$

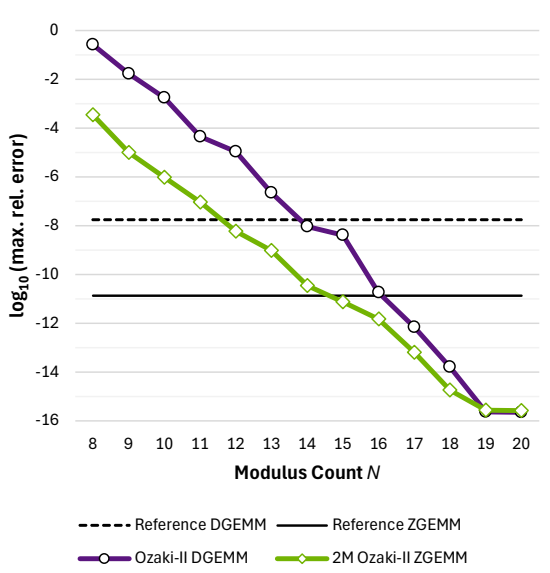
In particular $C_- = C_+^T$, so we only need to compute a single real multiplication C_+ for each m, providing the expected 2x speedup while still computing full GEMMs.

`HER2K`: Apply 2M to each multiplication. If we use the same scaling factors for A and B (i.e. $\sigma = \tau$), we can bring the matrix addition into the residue domains:

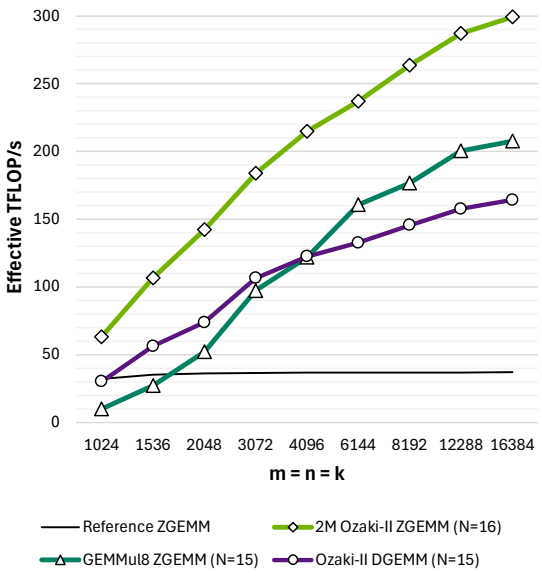
$$C_+ = (A_r + sA_i)(B_r - sB_i)^T + (B_r + sB_i)(A_r - sA_i)^T \bmod m,$$
$$C_- = (A_r - sA_i)(B_r + sB_i)^T + (B_r - sB_i)(A_r + sA_i)^T \bmod m.$$

Again $C_- = C_+^T$, reducing four real multiplications (per modulus) to two.

[3]The author thanks Robert Parrish for pointing out this method of reducing SYRK to 2M GEMM.



(a) Accuracy as a function of modulus count, N, measured as the maximum relative error over all entries of the product matrix C.



(b) Performance at varying square sizes, with fixed precision ($\log_2 M \approx 117$). TFLOP/s values are calculated using the number of floating-point operations required by reference GEMM.

Figure 1 – Performance and accuracy of 2M ZGEMM. All results measured on an NVIDIA B200 GPU. Reference implementation: `cublasZgemm` from cuBLAS 13.1, with emulation disabled.

SYRK, SYR2K: We convert (real) SYRK to an $n \times (k/2)$ HERK. Assuming for simplicity k is even, split A evenly in the k dimension: $A = \left[\begin{array}{c|c} A_1 & A_2 \end{array}\right]$ and form $Z = A_1 + iA_2$. Then

$$ZZ^H = (A_1A_1^T + A_2A_2^T) + i(A_2A_1^T - A_1A_2^T) = AA^T + i(A_2A_1^T - A_1A_2^T).$$

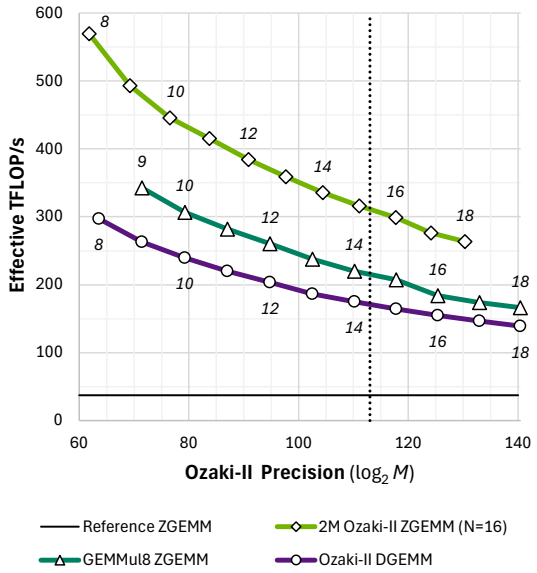
Hence $AA^T = \mathrm{Re}(ZZ^H)$, which we compute using 2M HERK. The same method converts SYR2K to a HER2K problem of half the k dimension.

6 Computational Results

To demonstrate the efficacy of the 2M method, we present in Figure 1 performance and accuracy results for int8-emulated 2M ZGEMM on a single NVIDIA B200 GPU. All results are with $\alpha = 1$, $\beta = 0$, and use the Gaussian integer version of 2M.

Figure 1(a) confirms that 2M ZGEMM reaches the expected accuracy results for Ozaki-II. We measure accuracy as the maximum relative error over each entry of C, on a test case with $m = n = k = 4096$, while varying the number of moduli N. Input matrices are chosen using Uchino et al.'s data distribution [5] with $\phi = 0.5$. Overall, convergence matches the overall trend of real Ozaki-II DGEMM. Note that the choice of complex multiplication method (2M, 3M, or 4M) does not affect accuracy as multiplications are always performed in exact integer arithmetic, and for this reason we do not undertake a detailed accuracy study here, instead referring the reader to [3–6].

Figure 1(b) shows 2M ZGEMM performance across a range of square problem sizes, using $N = 16$ moduli. Performance is measured in effective teraflops per second, calculated using the floating point operation count from the baseline algorithm: $8mnk$ operations for ZGEMM,



(a) Performance vs. precision, $m = n = k = 16384$. Marked data points show modulus count N. The dotted vertical line marks the approximate precision of reference floating-point GEMM.

m = n = k	8	10	12	14	16	18
1024	3.1	2.2	2.1	2	2	1.7
1536	4.7	3.5	3.3	3.1	3	2.5
2048	6.3	4.8	4.5	4.1	3.9	3.3
3072	8.7	6.7	6.1	5.4	5	4.3
4096	10.1	7.9	7.1	6.2	5.8	5
6144	11.7	9.1	8	7.1	6.4	5.6
8192	13.2	10.4	9	8	7.2	6.3
12288	14.7	11.5	10	8.7	7.8	6.8
16384	15.4	12	10.4	9.1	8.1	7.1

Modulus Count N

(b) Ozaki-II 2M ZGEMM speedups over reference ZGEMM.

Figure 2 – Additional 2M ZGEMM performance data.

$2mnk$ for DGEMM. The modulus count N is chosen to achieve accuracy similar to native-floating point ZGEMM; in fact, Ozaki-II 2M ZGEMM is approximately 10× more accurate than native floating-point ZGEMM for this input data (cf. Figure 1(a)).

With $N = 16$, 2M ZGEMM is up to 8× faster than reference 4M fp64 ZGEMM (cuBLAS 13.1), and 1.4–1.5× faster than Uchino et al.'s 3M-based `GEMMul8` library.[4] 2M also achieves an average 1.84× speedup in effective TF/s compared to Ozaki-II DGEMM at the same accuracy. That is, in absolute time a single ZGEMM costs approximately $4/1.84 \approx 2.16$ DGEMMs, close to theoretical expectations. The cost is not exactly 2 DGEMMs primarily because 2M requires slightly more moduli than the other Ozaki-II methods (16 vs. 15) to achieve a similar accuracy level. Again, this is due to the restricted set of moduli available for 2M.

In **Figure 2(a)** we fix the problem size to $m = n = k = 16384$ and vary N to show the tradeoff between performance and accuracy. 2M shows similar speedups over `GEMMul8`'s 3M ZGEMM and Ozaki-II DGEMM across the range of N, with performance up to 570 TF/s (15× reference ZGEMM) at $N = 8$. **Figure 2(b)** provides a more detailed heatmap of speedups over reference ZGEMM at a range of sizes and accuracy settings.

7 Conclusion

With each successive generation of CPU and GPU hardware, narrow-precision GEMM throughput continues to outpace traditional single and double precision arithmetic. The Ozaki-II scheme and other emulation methods take advantage of this trend, using integer quantization and small integer GEMMs to accelerate floating-point GEMMs.

[4] Tested version: `https://github.com/RIKEN-RCCS/GEMMul8/commit/3115e709`

In this paper, we have shown that this integer-quantized approach opens up new algorithms for complex GEMMs that leverage the algebraic structure of quadratic integer rings. In particular, 2M multiplication reduces a complex GEMM to two real GEMMs of the same size, instead of the three or four real GEMMs required by previous algorithms. For rank-k updates, 2M reduces a complex rank-k update ($C = AA^H$) to a real GEMM of the same, and a real rank-k update ($C = AA^T$) to a real GEMM of half the size. The new 2M algorithm is easy to implement, and achieves practical speedups that closely match these theoretical predictions.

Acknowledgements

The author is grateful for helpful reviews from numerous colleagues, which greatly improved this paper.

References

[1] K. Goto and R. Van De Geijn, *High-performance implementation of the level-3 BLAS*, ACM Transactions on Mathematical Software (TOMS), 35 (2008), pp. 1–14.

[2] N. J. Higham, *Stability of a method for multiplying complex matrices with three real matrix multiplications*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 681–687.

[3] A. Schwarz, A. Anders, C. Brower, H. Bayraktar, J. Gunnels, K. Clark, R. G. Xu, S. Rodriguez, S. Cayrols, P. Tabaszewski, and V. Podlozhnyuk, *Guaranteed DGEMM accuracy while using reduced precision tensor cores through extensions of the Ozaki scheme*, in Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region, SCA/HPCAsia '26, New York, NY, USA, 2026, Association for Computing Machinery, pp. 91–101.

[4] Y. Uchino, Q. Ma, T. Imamura, K. Ozaki, and P. L. Gutsche, *Emulation of complex matrix multiplication based on the Chinese remainder theorem*, 2025.

[5] Y. Uchino, K. Ozaki, and T. Imamura, *High-performance and power-efficient emulation of matrix multiplication using INT8 matrix engines*, in Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC Workshops '25, ACM, Nov. 2025, pp. 1824–1831.

[6] Y. Uchino, K. Ozaki, and T. Imamura, *Error analysis of matrix multiplication emulation using Ozaki-II scheme*, 2026.

[7] F. G. Van Zee and T. M. Smith, *Inducing complex matrix multiplication via the 3m and 4m methods*, FLAME Working Note 81, University of Texas at Austin, 2016.

[8] S. Winograd, *On multiplication of 2×2 matrices*, Linear algebra and its applications, (1971).