Performance Evaluation of Accelerated Complex Multiple-Precision LU Decomposition

Tomonori Kouya(✉)

Shizuoka Institute of Science and Technology, 2022-2 Toyosawa, Fukuroi, Shizuoka 437-8555, Japan
kouya.tomonori@sist.ac.jp
https://www.sist.ac.jp

Abstract. The direct method is one of the most important algorithms for solving linear systems of equations, with LU decomposition depleting a significant portion of the computation time. This study explores strategies for accelerating complex LU decomposition, using multiple-precision floating-point arithmetic of the multiple-component type. Specifically, the potential efficiency gains are explored using a combination of single instruction and multiple data stream (SIMD) and the 3M method for complex matrix multiplication. This approach was compared with the implementation of the direct method in MPLAPACK, using benchmark tests focusing on computation time and numerical errors.

Keywords: LU decomposition · multiple-precision floating-point arithmetic · complex arithmetic · parallelization

1 Introduction

Linear systems of equations are frequently encountered in scientific computations; therefore, various solvers have been developed to address the computational environment and mathematical properties of these equations. The direct method, including LU decomposition, is considered standard because it can solve equations with a limited number of algebraic operations, maintaining numerical stability using a pivoting strategy. The high performance linpack (HPL) benchmark test [9] employs the direct method to select the top 500 supercomputers.

In recent years, considerable research has been conducted on mixed-precision computing, thereby tailoring the number of precision digits (the length of the mantissa) of the floating-point (FP) numbers to the requirements of the problem. This approach improves computational performance by combining various types of hardware-oriented IEEE754 FP arithmetic, such as binary16 (11-bit precision), binary32 (24-bit), and binary64 (53-bit). However, the inadequate accuracy of the calculation poses a significant disadvantage. Thus, there is an

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024 Workshops, LNCS 14818, pp. 3–19, 2024.
https://doi.org/10.1007/978-3-031-65273-8_1

increasing demand for "reproducible computing" to minimize the environmental dependence of the calculation results. For well-conditioned problems, speed-up should be pursued using fast binary 16, 32, and 64 FP operations with relatively short precision and hardware-oriented computation. Conversely, for ill conditioned problems, multiple-precision FP operations exceeding binary64 should be employed to ensure accuracy and reproducibility. In particular, stable solutions to nonlinear problems based on linear computation, require minimizing the rounding errors in the intermediate results. Utilizing a standard multiple-precision FP arithmetic library that is known for user-friendly interface and high performance, such as QD [1] and multiple-precision floating-point reliable (MPFR) [10], is essential. These libraries are incorporated into MPLAPACK/MPBLAS [7], serving as a de facto standard multiple-precision linear computation environment. We developed three high-performance multiple-precision real basic linear subprograms (BLAS) of double-double (DD, 106 bits), triple-double (TD, 159 bits), and quadruple-double (QD, 212 bits) arithmetic using advanced vector extensions two (AVX2) [5]. The results have been compiled into BNCmatmul [4] and released as open-source software.

Based on these results, this paper discusses the performance evaluation of complex multiple-precision LU decomposition, particularly focusing on multi-component DD, TD, and QD precision, further employing 3M, a type of complex multiplication method. Benchmark tests were conducted to compare the acceleration provided by complex arithmetic using AVX2, LU decomposition using the Strassen algorithm [11] and Ozaki scheme [8], and parallelization using OpenMP.

2 Acceleration of Complex Basic Linear Computation

As mentioned above, we implemented real BLAS and LU decomposition in purely real values using single instruction and multiple data stream (SIMD) operations such as AVX2 and a library of multiple-precision linear computations that supported parallelization using open multi-processing (OpenMP). Additionally, we developed arbitrary precision floating-point operations based on MPFR and multi-component fixed-precision operations supporting DD, TD, and QD precision, which are faster than MPLAPACK/MPBLAS. Building on this foundation, we implemented a complex BLAS [6]. Notably, despite publishing a collection of results on arbitrary-precision complex LU decomposition [6], we have yet to parallelize it. In the following section, strategies for improving the performance of complex-valued linear computation, which serves as the basis for complex LU decomposition, are explained.

2.1 3M Method for Complex Linear Computation

For any $a, b \in \mathbb{C}$, the standard complex multiplication $ab \in \mathbb{C}$ is calculated by using the 4M (multiplication) method, which involves four real multiplications:

$$\begin{aligned} ab := &(\mathrm{Re}(a)\mathrm{Re}(b) - \mathrm{Im}(a)\mathrm{Im}(b)) + \{\mathrm{Im}(a)\mathrm{Re}(b) \\ &+\mathrm{Re}(a)\mathrm{Im}(b)\} \cdot \mathrm{i}. \end{aligned} \tag{1}$$

In contrast, the 3M method is executed as follows:

$$\begin{aligned} t_1 &:= \mathrm{Re}(a)\mathrm{Re}(b),\ t_2 := \mathrm{Im}(a)\mathrm{Im}(b) \in \mathbb{R} \\ ab &:= (t_1 - t_2) + \{(\mathrm{Re}(a) + \mathrm{Im}(a)) \\ &\quad (\mathrm{Re}(b) + \mathrm{Im}(b)) - t_1 - t_2\} \cdot \mathrm{i} \in \mathbb{C}. \end{aligned} \tag{2}$$

The 3M method retains the same computation in the real part as the 4M method but reduces the number of real multiplications in the imaginary part and hence its name. The efficiency of the 3M method cannot be ascertained without benchmark testing of its implementation because it includes three additional additions compared with the 4M method. Nevertheless, The 3M method has proven to be valuable for tasks such as matrix multiplication [12] and arbitrary-precision complex arithmetic [2,3], particularly in scenarios with extensive computation times dedicated to multiplication.

We implemented a multiple-precision matrix multiplication based on the 3M method such that for any $A \in \mathbb{C}^{m \times n}$ and $B \in \mathbb{C}^{n \times l}$,

$$\begin{aligned} T_1 &:= \mathrm{Re}(A)\mathrm{Re}(B),\ T_2 := \mathrm{Im}(A)\mathrm{Im}(B) \in \mathbb{R}^{m \times l} \\ AB &:= (T_1 - T_2) + \{(\mathrm{Re}(A) + \mathrm{Im}(A)) \\ &\quad (\mathrm{Re}(B) + \mathrm{Im}(B)) - T_1 - T_2\} \cdot \mathrm{i} \in \mathbb{C}^{m \times l}. \end{aligned} \tag{3}$$

This approach allows for complex matrix multiplication based on existing high-performance real matrix multiplication techniques. Although the efficiency of this implementation remains uncertain because of the three additional matrix additions and subtractions, benchmark tests are necessary to evaluate the performance of the multiple-precision complex matrix multiplication.

2.2 Acceleration Using AVX2 SIMDization

The data structures for the implemented multiple-precision complex vectors and matrices are shown in Fig. 1.

As described above, the complex basic linear computation, which included the 3M method, was constructed using a real multiple-precision BLAS developed through SIMDization, such as AVX2 in our library. Complex vectors and matrices were allocated as separate arrays for the real and imaginary parts of multi-component multiple-precision FP data types, as depicted on the right of Fig. 1. However, as depicted on the left of Fig. 1 for arbitrary-precision FP arithmetic, the same data type (a set of real and imaginary parts) in multple-precision complex arithmetic (MPC) [3] was adopted for a single element. The MPC based on MPFR employed the 3M method for element-wise complex multiplication to ensure high speed.

This approach simplifies the implementation of fast complex matrix multiplication based on the formula in (3) for multi-component-type multiple-precision calculations. Having already implemented four matrix multiplication algorithms: simple matrix multiplication using triple loops, block matrix multiplication,

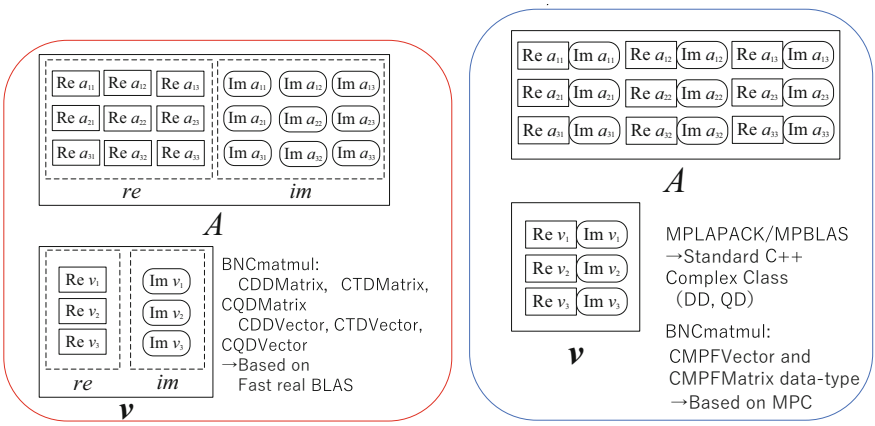


Fig. 1. Two types of data structure for multiple-precision complex vectors $\mathbf{v} \in \mathbb{C}^3$ and matrices $A \in \mathbb{C}^{3\times 3}$: multi-componet-type multiple-precision (left) and arbitrary precision standard C++ complex class used in MPBLAS (right)

Strassen matrix multiplication (and the Winograd algorithm), and the Ozaki scheme, complex matrix multiplication was achieved by combining them.

Therefore, SIMDization of complex arithmetic units is not necessary for basic matrix multiplication. However, to speed up the complex LU decomposition, we decided to SIMDize the arithmetic, similar to real LU decomposition, to speed up the element-by-element forward elimination calculation. For multi-component complex vectors and matrices, SIMD Load/Store instructions can be used to retrieve and store elements because the real and imaginary parts are separated. Using this approach, complex DD operations using AVX2 (`_bncavx2_rcdd_add`, `sub`, `mul`, `div`, `inv`(reciprocal)) can be implemented as shown in Fig. 2. SIMDized complex TD and QD operations are implemented in the same way.

The SIMDized complex arithmetic method was based on a combination of SIMDized real arithmetic methods.

2.3 Parallelization with OpenMP

For parallelization, the multi-component multiple-precision complex linear computations were based on real linear computation units. Real matrix multiplication based on the Strassen algorithm was parallelized in sections, as illustrated in Fig. 3. The nested nature of parallelization precluded performance improvement beyond eight threads. Arbitrary-precision complex matrix multiplication was implemented using MPC operations, including parallelization.

The Ozaki scheme divides the computation into shorter length elements to avoid introducing rounding errors into the short-precision matrix multiplication base of general matrix multiplications (GEMM), specifically in this case, xGEMM using binary64 DGEMM. The product obtained by xGEMM was then combined with multiple-precision matrix addition to obtain a high-precision

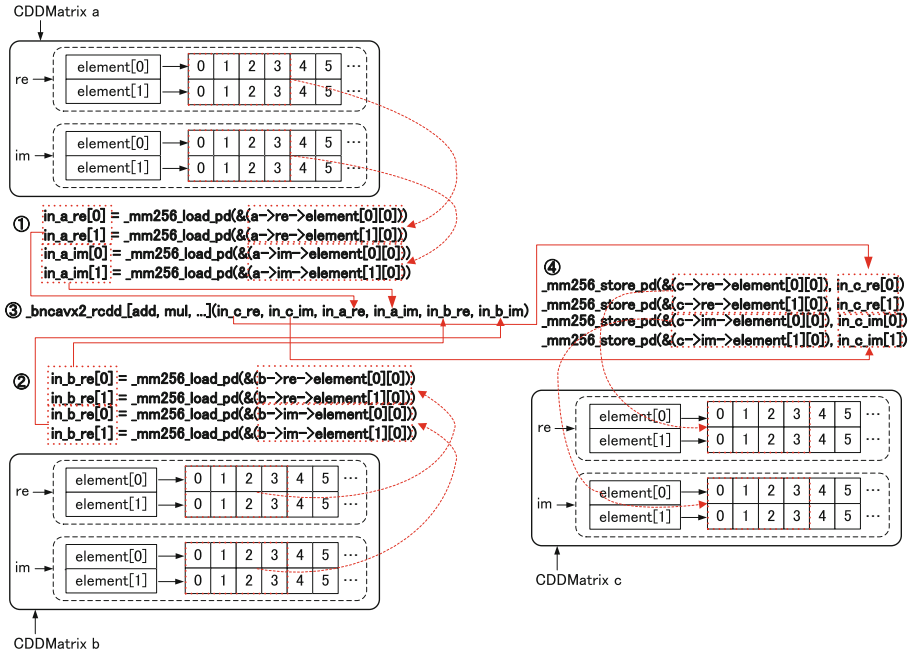


Fig. 2. Complex double-double arithmetic with AVX2 utilized in complex BLAS and LU decomposition

matrix product. To parallelize this process, parallelized xGEMMs are used or OpenMP loop parallelization can be performed for each xGEMM, as illustrated in Fig. 4. In this study, the latter approach was used for parallelization.

2.4 Benchmark Test for Complex Matrix Multiplication

The AMD EPYC environment was used to present the results obtained from the benchmark tests.

CPU and OS AMD EPYC 9354P 3.7 GHz 32 cores, Ubuntu 20.04.6 LTS
C/C++ Intel Compiler version 2021.10.0
Compiler Option `-O3 -std=c++11 -fp-model precise`
with AVX2 `-axCORE-AVX2 -march=skylake -mtune=skylake -mcpu=skylake`
MPLAPACK 2.0.1, GNU MP 6.2.1, MPFR 4.1.0, MPC 1.2.1

In addressing the problem of computing the matrix product $C := AB$ for the given complex square matrices A and $B \in \mathbb{C}^{n \times n}$, the elements of A and B used for benchmarking were uniformly distributed random numbers in the interval $[0, 1]$. For the Ozaki scheme, to ensure sufficient accuracy, a maximum of six divisions were used for DD accuracy, eight divisions for TD accuracy, and 12 divisions for QD accuracy.

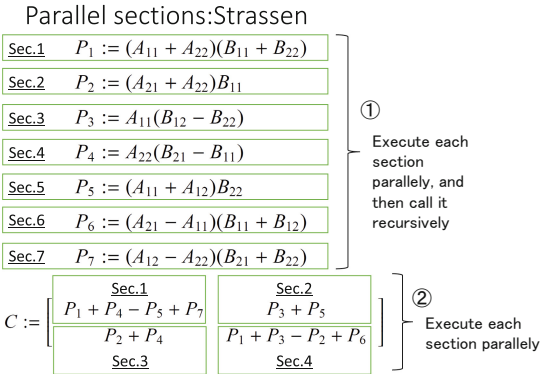


Fig. 3. Parallelized Strassen algorithm

For all the benchmark tests used in this study, the block size of the matrix multiplication was fixed at 32×32, and 32 was chosen as the minimum number of matrix rows and columns when stopping the recursive loop in the Strassen algorithm. The Ozaki scheme used the DGEMM of Intel math kernel library banded in the Intel compiler kit.

The results of the DD-precision matrix multiplication are presented in Fig. 5. The left figure shows the computation time in seconds, and the right figure shows the corresponding speed-up ratio achieved by parallelization using OpenMP. In the left figure, our implementation of the Strassen and Ozaki schemes achieved up to 100 times performance improvement compared to the CGEMM routine of MPBLAS in serial calculations. Additionally, the 3M method exhibited a speed improvement of approximately 1.3 times over the 4M method, whereas the Strassen method achieved a speed-up of 3x-4x. The computation time of the Ozaki scheme was nearly equivalent to that of the AVX2 Strassen scheme because of the small number of partitions, whereas the block matrix multiplication based on the 3M method was slightly slower than that based on the Strassen matrix multiplication.

In OpenMP parallelization, the Strassen matrix multiplication did not exhibit any acceleration, even when 32 threads were used. Similarly, the Ozaki scheme demonstrated only a modest 2x speed-up. In contrast, block matrix multiplication achieved a significant improvement, with speeds of up to 12 times faster.

Next, the results obtained by TD-precision matrix multiplication are presented in Fig. 6.

The AVX2 SIMDized TD-precision Strassen block matrix multiplication was slower than that of the non-SIMDized version although the reason for this discrepancy remains unclear. Therefore, only non-SIMDized multiplications were employed in TD precision. The 3M method demonstrated a speed improvement of approximately 1.3 times compared to the 4M method, which is common with DD precision. The speed-up achieved by the Ozaki scheme was approximately eight times faster than that of the Strassen 3M method.

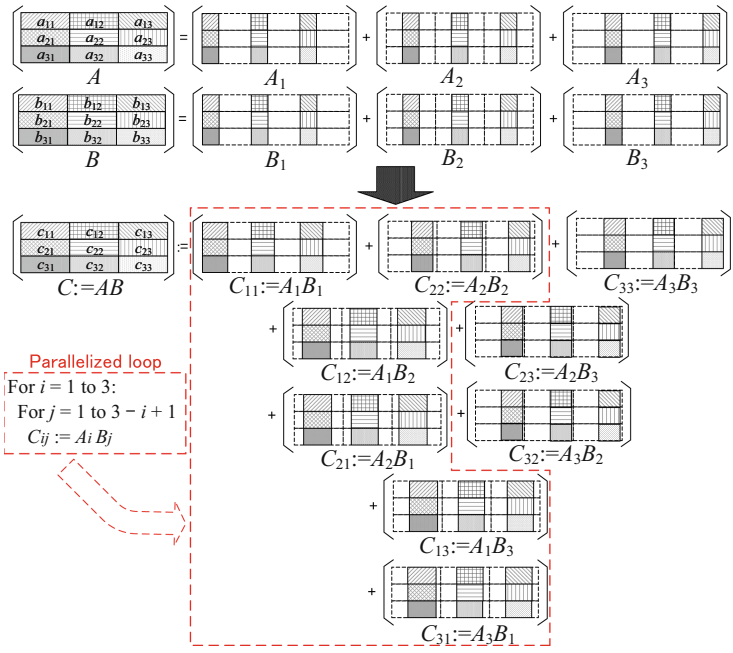


Fig. 4. Parallelized Ozaki scheme

In OpenMP parallelization, the Strassen matrix multiplication was observed to be slower, and the Ozaki scheme was only approximately twice as fast, even when utilizing 32 threads, as in DD accuracy. Block matrix multiplication was up to 20 times faster.

Finally, the results of QD-precision matrix multiplication are shown in Fig. 7.

Similar to DD and TD precision, the Strassen matrix multiplication was approximately twice faster in performance using AVX2 optimization and 1.3 times faster in performance using the 3M method.

In OpenMP parallelization, the Strassen matrix multiplication demonstrated faster performance, whereas the Ozaki scheme achieved only a speed-up of 2x compared to the DD- and TD-precision schemes despite utilizing 32 threads. Block matrix multiplication was 20 times faster than TD precision.

These results demonstrate that while the 3M method can speed up serial complex matrix multiplication, it does not speed up parallelization in relation to the number of threads, except in the case of block matrix multiplication.

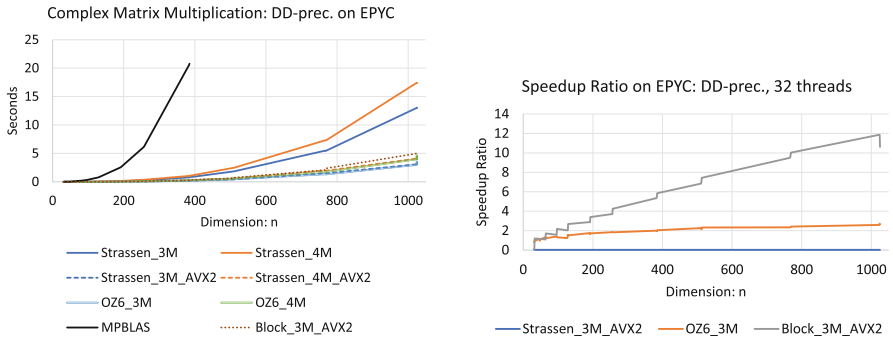


Fig. 5. DD-precision: computation time in seconds (left) and speed-up ratio when using 32 threads (right)

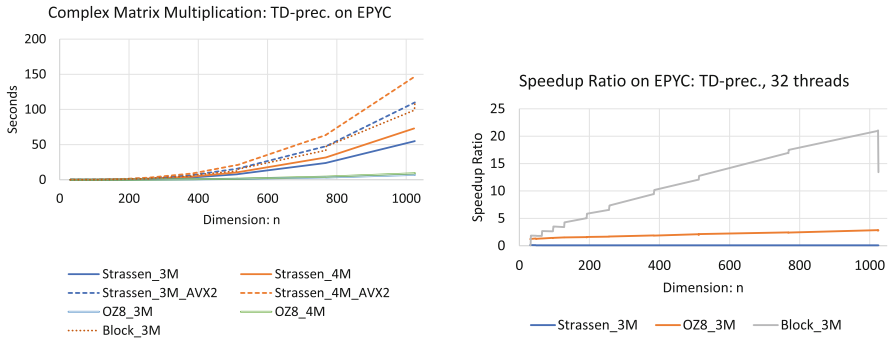


Fig. 6. TD-prec. : Computation time in second (left) and speed-up ratio when using 32 threads (right)

3 Acceleration of Multiple-Precision Complex LU Decomposition

Complex LU decomposition was implemented based on a complex basic linear computation that integrates all the implemented speed-up techniques (3M method, SIMD, and OpenMP parallelization) to achieve high performance. The following is a concise description of the implementation:

The target n-dimensional complex linear system is expressed as

$$A\mathbf{x} = \mathbf{b}, \tag{4}$$

where $A \in \mathbb{C}^{n\times n}$, $\mathbf{x} \in \mathbb{C}^n$, and $\mathbf{b} \in \mathbb{C}^n$. The (4) was solved to determine the unknown vector $\mathbf{x}$ given A and b.

The current standard implementation of the LU decomposition uses fast matrix multiplication. As shown in Fig. 8, a certain block size $K(=$ NDIM) is set in advance, and $A - L_{21}U_{12}$ is performed to compute A_{22}. Consequently, the

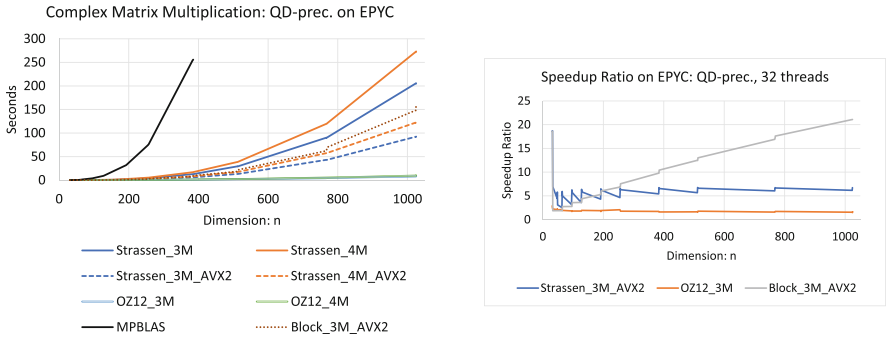


Fig. 7. QD-precision: Computation time in seconds (left) and speed-up ratio when using 32 threads (right)

computational performance varies depending on the value of K. The LU decomposition (Cgetrf function) of MPLAPACK in the comparisons, also used complex matrix multiplication (CGEMM function). In all cases, LU decomposition was computed using a partial pivoting strategy.

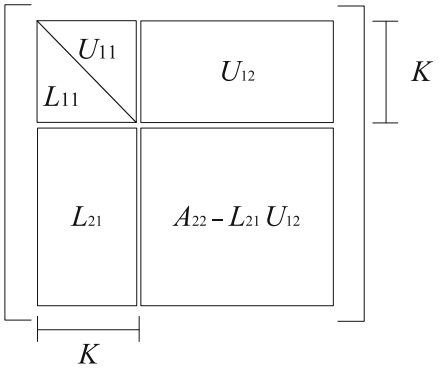


Fig. 8. LU decomposition using matrix multiplication

In addition, a standard LU decomposition approach (referred to as "Normal LU" or "Normal"), which eliminates rows sequentially without updating or performing matrix multiplication, was implemented using complex AVX2 arithmetic, as illustrated in Fig. 2, to benefit from the ease of speeding up the process using SIMD instructions. AVX2 was effective in real LU decomposition, with the Ozaki scheme emerging as the fastest. However, the question remains: will the same effect be obtained for complex LU decomposition?

4 Benchmark Test of Complex LU Decomposition

During the benchmark test, the computation time and maximum error of the numerical solution of the LU decomposition were measured by varying K while calculating DD, TD, and QD precision. The graphs in Fig. 9 to Fig. 17 show the computation time for normal LU and LU decomposition using matrix multiplication, as well as the relative error of the numerical solutions obtained through forward and backward substitutions.

We focused on solving the n-dimensional complex linear system depicted in Equation (4). For benchmarking purposes, the elements of A were generated as uniform random numbers within the range $[0, 1]$.

The true solution $\mathbf{x}$ is $(\mathbf{x})_k = k + k \cdot \mathrm{i}$ and the constant vector were derived by computing $\mathbf{b} := A\mathbf{x}$ using 2048-bit precision calculation.

4.1 DD-Precision Computation

First, we present the results of the DD accuracy calculation, which involved measuring the relative error and computation time while increasing $1 \le K \le n$ by 32.

In examining the relative error (Fig. 9), the MPLAPACK and AVX2 SIMDized normal LU decompositions demonstrated superior performance. The normal LU and LU decompositions using the Ozaki scheme and Strassen matrix multiplication, respectively, yielded similar results. Overall, the results show that the accuracy tends to decrease when Strassen is used. In this problem, the Ozaki scheme with DGEMM and five divisions did not achieve optimal accuracy, necessitating more than six divisions. Therefore, the results are presented for five, six, and seven divisions.

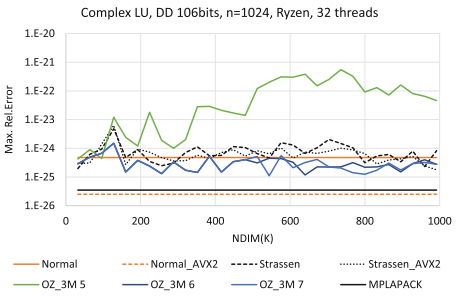


Fig. 9. Complex LU decomposition with DD-precision: Max. relative error

Figure 10 demonstrates the measured serial computation time (left) and 32-thread parallel computation time (right). The results with normal LU decomposition and AVX2 acceleration are represented by the solid orange and dashed lines, respectively. The relative error of MPLAPACK is depicted as a solid black

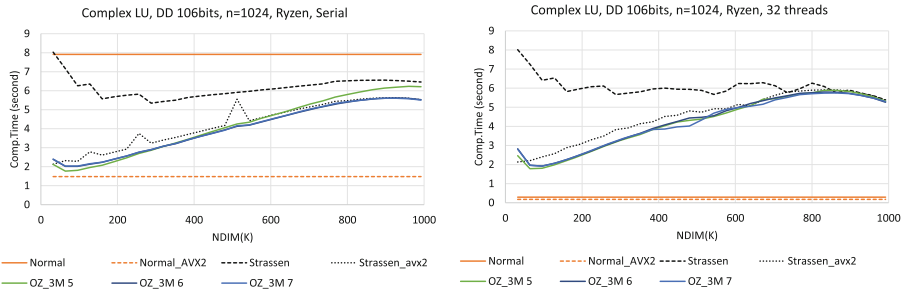
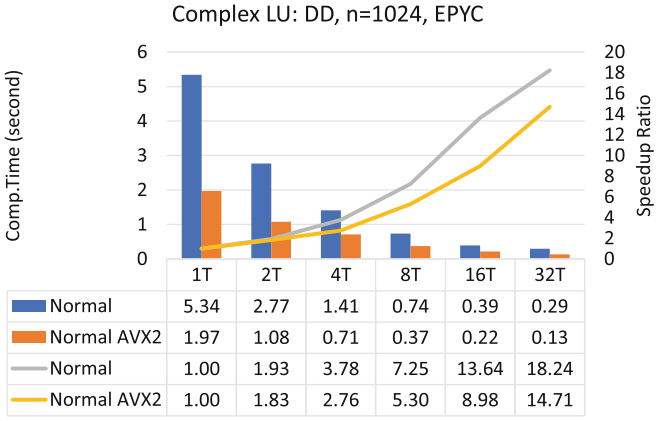


Fig. 10. Complex LU decomposition with DD-precision: serial computation in seconds (left), 32-thread computation in seconds (right)

line although it was not included in the computation time graph because of the significant difference in scale (130.7 s).

The AVX2 normal LU decomposition exhibited the fastest serial computation, whereas the AVX2 Strassen LU decomposition and Ozaki scheme showed similar computation times, with the smaller K values requiring less time. However, they were not as fast as the AVX2 normal LU decomposition.

In the case of DD-precision calculations, OpenMP parallelization of matrix multiplication initially increased the computation time with two threads, after which it became faster in proportion to the number of threads. Therefore, with 32 threads, the computation time was approximately the same as that for serial computation, and as a result, the parallelization effect was not enhanced. However, as depicted in Fig. 11, the normal LU decomposition achieved a speed-up in proportion to the number of threads, and the AVX2 version remained the fastest, even with 32 threads.



	1T	2T	4T	8T	16T	32T
Normal	5.34	2.77	1.41	0.74	0.39	0.29
Normal AVX2	1.97	1.08	0.71	0.37	0.22	0.13
Normal	1.00	1.93	3.78	7.25	13.64	18.24
Normal AVX2	1.00	1.83	2.76	5.30	8.98	14.71

Fig. 11. Parallelization of normal complex LU decomposition: DD-precision

The reason for this phenomenon is believed to be the higher computational complexity inherent in complex LU decomposition compared with real LU decomposition despite the enhanced performance of complex operations achieved through SIMD. As previously mentioned, although SIMD algorithms are not essential for complex basic linear computation implementations, they may facilitate acceleration of the Fast Fourier Transform, which is required locally for LU decomposition.

4.2 QD-Precision Computing

Next, the results of QD accuracy calculations are presented.

Figure 12 displays the relative errors obtained from QD-precision benchmark tests. As shown in this figure, MPLAPACK slightly outperforms the AVX2 SIMDized normal LU decomposition. LU decomposition using the Ozaki scheme and Strassen matrix multiplication exhibit larger relative errors than normal LU decomposition.

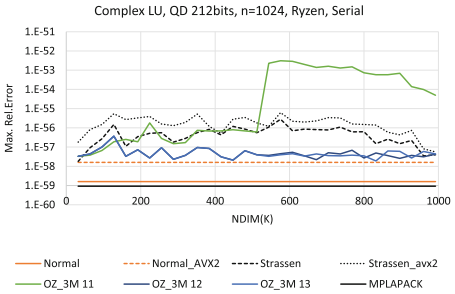


Fig. 12. Complex LU decomposition with QD precision: Max. relative error

Figure 13 illustrates the computation time obtained from QD-precision benchmark tests. The graph shows the measured serial computation time (left) and 32-thread parallel computation time (right) using the standard LU decomposition and AVX2-accelerated approach, represented by solid and dashed orange lines, respectively. For this particular problem, the Ozaki scheme using DGEMM does not achieve the highest QD accuracy with 11 divisions and requires more than 12 divisions; hence, the results for 11, 12, and 13 divisions are displayed. The relative error for MPLAPACK is represented by the solid black line; however, the serial computation time is 595.4 s; therefore, it is not included in the graph of the computation time because it is on a different scale.

For serial calculations, AVX2 SIMDized normal LU decomposition generally exhibited the fastest performance. However, the Ozaki LU decomposition scheme was slightly faster when K was small. The Strassen LU decomposition, even with AVX2 SIMDized, was not as fast as the AVX2 SIMDized normal LU decomposition.

In QD-precision calculations, OpenMP parallelization of matrix multiplication accelerated the computation linearly with the number of threads. However,

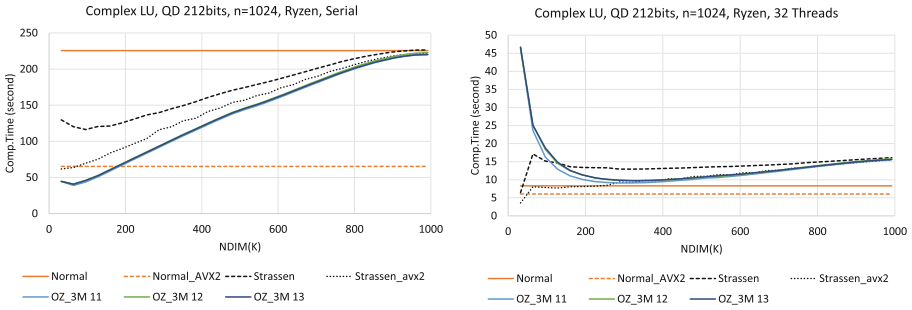


Fig. 13. Complex LU decomposition in QD precision: serial computation in seconds (left), 32-thread computation in seconds (right)

the parallelization effect of the normal LU decomposition surpassed this, with the AVX2 SIMDized normal LU decomposition remaining the fastest overall, except in certain cases where it was outperformed by AVX2 SIMDized Strassen LU decomposition. As illustrated in Fig. 14, although parallelization efficiency is high, AVX2 slows it down.

4.3 TD-Precision Computing

Finally, we present the results of the TD-precision calculation, which represents an intermediate accuracy level between DD and QD.

In this case, as depicted in Fig. 15, for TD accuracy, the Ozaki scheme with DGEMM did not achieve the highest accuracy with seven divisions, requiring eight or more divisions.

The relative error analysis revealed that similar to the case of QD accuracy, the AVX2 SIMDized normal LU decomposition performed the best. Further-

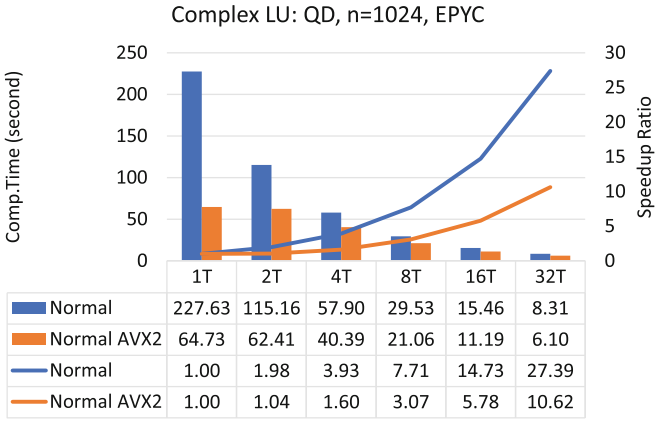


Fig. 14. Parallelization of normal complex LU decomposition: QD-precision

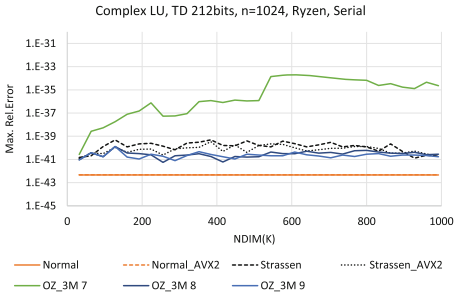


Fig. 15. TD-precision complex LU decomposition: Max. relative error

more, the relative errors observed in the LU decomposition using the Ozaki scheme and Strassen matrix multiplication were greater than that of normal LU decomposition.

A graph depicting the computation time is presented in Fig. 16, where the 32-thread parallelization using TD accuracy was omitted because of the significant increase in computation time for small K.

In serial calculations, the AVX2 SIMDized normal LU decomposition in general did not outperform the others, with the LU decomposition using the Ozaki scheme being significantly faster for small K values. In TD-precision calculations, OpenMP parallelization of matrix multiplication displayed a straightforward speed-up proportional to the number of threads. However, the parallelization effect of the normal LU decomposition outweighed this effect. Normal LU decomposition with AVX2 remained the fastest, except in the case of LU decomposition with the Ozaki scheme performing better. As shown in Fig. 17, the parallelization efficiency is considerable; however, the ratio is reduced compared to DD and QD normal LU decomposition. The reason the parallel efficiency of LU decomposition with TD precision is lower than that of DD and QD is unclear; however, the fact that the precision length is not a power of 2 for binary64 may have hindered compiler optimization.

4.4 Summary of High-Performance Complex LU Decomposition

These benchmark results indicate that the AVX2 SIMDized normal LU decomposition generally exhibits the fastest performance in terms of both DD and QD accuracy with good parallelization. A summary focusing solely on the computation time is presented in Table 1, highlighting the minimum computation time achieved for Strassen with AVX2 for DD and QD, and without AVX2 in TD.

Finally, the 32-thread AVX2 normal LU decomposition was demonstrated to outperform MPLAPACK with serial computations by approximately 726 times in DD accuracy and approximately 91 times in QD accuracy. These findings suggest that the default setting for complex LU decomposition should be the AVX2 SIMDized normal LU decomposition.

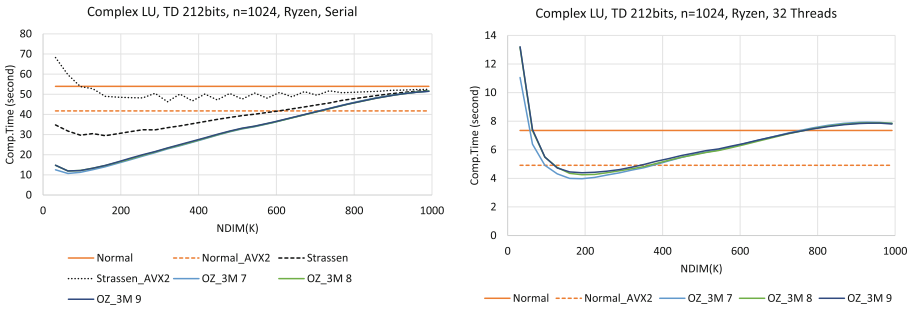
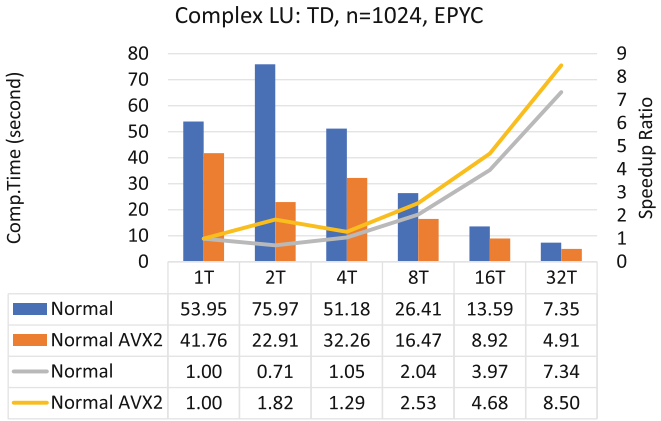


Fig. 16. Complex LU decomposition with TD-precision: serial computation in seconds (left) and 32-thread computation in seconds (right)



	1T	2T	4T	8T	16T	32T
Normal	53.95	75.97	51.18	26.41	13.59	7.35
Normal AVX2	41.76	22.91	32.26	16.47	8.92	4.91
Normal	1.00	0.71	1.05	2.04	3.97	7.34
Normal AVX2	1.00	1.82	1.29	2.53	4.68	8.50

Fig. 17. Parallelization of normal complex LU decomposition with TD-precision

Table 1. Minimum computation time of complex LU decomposition in second

DD precision, $n = 1024$					
	MPLAPACK	Normal	Normal AVX2	Strassen AVX2	OZ
Serial	130.7	5.3	2.0	2.1 ($K = 32$)	2.0 ($K = 64$)
Parallel(32T)	–	0.29	0.13	2.1 ($K = 32$)	1.9 ($K = 96$)
TD precision, $n = 1024$					
Serial	–	54.0	41.8	46.7 ($K = 384$)	11.9($K = 64$)
Parallel(32T)	–	7.4	4.9	8.5 ($K = 992$)	4.3($K = 192$)
QD precision, $n = 1024$					
Serial	595.4	227.6	64.7	60.1($K = 32$)	40.7($K = 64$)
Parallel(32T)	–	8.3	6.1	3.6($K = 32$)	9.8($K = 352$)

5 Conclusion and Future Works

The implementation of the multiple-precision complex basic linear computation based on the enhanced multiple-precision real basic linear computation demonstrated improved speed using the 3M method for matrix multiplication. Additionally, the complex LU decomposition was found to be sufficiently rapid through the utilization of SIMD complex operations and parallelization using OpenMP. However, the acceleration of LU decomposition was observed to be significant enough to surpass conventional LU decomposition algorithms in DD- and QD-precision, using the current complex matrix multiplication both in serial and parallel computations.

Future work will focus on enhancing the performance of numerical algorithms for ill-conditioned nonlinear problems by improving the following functions.

1. Implementation of a multiple-precision sparse matrix and extension of the current implementation to real and complex sparse matrices.
2. Acceleration of reproducible computations for binary64 and other types of precision arithmetic.
3. Extending fast multi-precision computation to the Python environment.

This includes seeking further speed-up of real and complex basic linear computations of dense matrices and LU decomposition. Of particular interest is confirming the potential speed-up of SIMD-enhanced multi-component multiple-precision real and complex arithmetic for other numerical algorithms.

Acknowledgment. This research was supported by JSPS Grant-in-Aid for Scientific Research 23K11127. We appreciate the sincere and important comments of the unknown reviewers who helped improve the overall presentation of this paper.

References

1. Bailey, D.: QD. https://www.davidhbailey.com/dhbsoftware/
2. Bini, D.A., Fiorentino, G.: Design, analysis, and implementation of a multiprecision polynomial rootfinder. Numer. Algorithms **23**, 127–173 (2000)
3. Enge, A., Théveny, P., Zimmermann, P.: MPC. http://www.multiprecision.org/mpc/
4. Kouya, T.: BNCmatmul. https://github.com/tkouya/bncmatmul
5. Kouya, T.: Acceleration of multiple-precision matrix multiplication based on multi-component floating-point arithmetic using AVX2. In: Gervasi, O., et al. (eds.) Computational Science and Its Applications - ICCSA 2021, pp. 202–217. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-86976-2_14
6. Kouya, T.: Acceleration of Complex Matrix Multiplication Using Arbitrary Precision Floating Point Arithmetic (2023)
7. MPLAPACK/MPBLAS: Multiple precision arithmetic LAPACK and BLAS. https://github.com/nakatamaho/mplapack
8. Ozaki, K., Ogita, T., Rump, S.M., Oishi, S.: Fast algorithms for floating-point interval matrix multiplication. J. Comput. Appl. Math. **236**, 1795–1814 (2012)

9. Petitet, A., Whaley, R.C., Dongarra, J., Cleary, A.: HPL - a portable implementation of the high-performance linpack benchmark for distributed-memory computers
10. Project, M.: The MPFR library. https://www.mpfr.org/
11. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**(4), 354–356 (1969). https://doi.org/10.1007/BF02165411
12. Van Zee, F.G., Smith, T.M.: Implementing high-performance complex matrix multiplication via the 3M and 4M methods. ACM Trans. Math. Softw. **44**(1), 1–36 (2017). https://doi.org/10.1145/3086466