

Acceleration of Multiple Precision Solver for Ill-Conditioned Algebraic Equations with Lower Precision Eigensolver

Tomonori Kouya(✉)

Shizuoka Institute of Science and Technology,
2200-2 Toyosawa, Fukuroi 437-8555, Japan
kouya.tomonori@sist.ac.jp

Abstract. There are some types of ill-conditioned algebraic equations that have difficulty in obtaining accurate roots and coefficients that must be expressed with a multiple precision floating-point number. When all their roots are simple, the problem solved via eigensolver (eigenvalue method) is well-conditioned if the corresponding companion matrix has its small condition number. However, directly solving them with Newton or simultaneous iteration methods (direct iterative methods) should be considered as ill-conditioned because of increasing density of its root distribution. Although a greater number of mantissa of floating-point arithmetic is necessary in the direct iterative method than eigenvalue method, the total computational costs cannot obviously be determined. In this study, we target Wilkinson's example and Chebyshev quadrature problem as examples of ill-conditioned algebraic equations, and demonstrate some concrete numerical results to prove that the direct iterative method can perform better than standard eigensolver.

Keywords: Multiple precision floating-point arithmetic · Algebraic equation · Eigensolver · Parallelization

1 Introduction

Currently, multiple precision floating-point (MPF) arithmetic are executed using reliable and highly performed de-facto standard libraries such as QD [2] and MPFR [14]. These have been developed since the end of the 20th century, and have been in use for over two decades. More convenient and efficient multiple precision numerical computation libraries such as MPLAPACK [10] and our BNCpack [6], are constructed on that of MPF libraries. Therefore, we can easily deal with various types of ill-conditioned problems in normal hardware and software environment for consumers.

We have published a paper [7] about deriving highly accurate abscissas of Gauss-type integration formulas using both "eigenvalue method" via symmetric tridiagonal matrix originated by Golub and Welsch [4] and "direct iterative

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13378, pp. 358–372, 2022.
https://doi.org/10.1007/978-3-031-10562-3_26

method" to directly calculate zeros of orthogonal polynomial using Newton iteration [17–19]. Our previous task with direct iterative method was to calculate the approximation of abscissas with user-required accuracy by combining binary64 eigensolver of LAPACK [8] and MPFR Newton iteration.

The performance of the eigenvalue method is slow but robust for effect by round-off error; hence, we can obtain good approximations even with binary64 arithmetic. However, the direct iterative method is well-performed with multiple precision arithmetic, but good initial guesses must be employed to guarantee its convergence. In our previous paper, we concluded that, to shorten total computational time, it is best to combine binary64 LAPACK eigensolver to derive initial guesses, and multiple precision Newton iteration to exploit a higher accuracy. From the current point of view, our proposed combination employs the "mixed precision" method for solving algebraic equations.

From these experiences, we have tried to combine multiple precision eigensolver of MPLAPACK, and the second- and third-order simultaneous iterative methods of BNCpack, such as the Durand-Kerner (DK) method, to solve two types of ill-conditioned algebraic equations with real and complex roots. One is well-known Wilkinson's problem [16], and another is calculation of abscissas of Chebyshev quadrature formula (Chebyshev quadrature problem for short), which has been experimented by Harumi Ono [12,13]. The algebraic equation of Chebyshev quadrature problem has already been studied in several published papers, but those papers are not known except in Japan because most of them are written only in Japanese. The only English non-referred paper in English can be found in Kokyoroku series of RIMS in Kyoto University [5]. Here, we briefly explain the previous results on the Chebyshev quadrature problem.

The algebraic equations derived from the Chebyshev quadrature formula have difficulties not only in calculating the accurate coefficients, but also in solving the equations with multiple precision floating-point arithmetic. Harumi Ono published papers on how to solve the 1024-th degree polynomial in 1979, and 2048-th degree in 1981 using the DK method with her original multiple precision arithmetic on Cray. Additionally, Masumoto et al. reported the numerical property of 20480-th degree coefficients when deriving them with MPF arithmetic [9] but did not achieve the obtained roots of those. We will describe the mathematical expression of the Chebyshev quadrature problem in Sect. 4.

In this study, we will relay the results obtained by combining DD Rgeev of MPLAPACK as a generator of initial guesses and the MPFR Durand-Kerner method from the point of view for mixed precision approach of acceleration.

The following two computational environments, EPYC and Xeon, are used in the rest of this study. MPLAPACK and our BNCpack, including QD and MPFR/GNU MP, are natively compiled with Intel Compiler.

EPYC AMD EPYC 7402P 24 cores, Ubuntu 18.04.6 LTS, GCC 7.5.0, Intel Compiler version 2021.4.0, MPLAPACK 1.0.1, BNCpack 0.8, MPFR 4.1.0

Xeon Intel Xeon W-2295 3.0 GHz 18 cores, Ubuntu 20.04.3 LTS, GCC 9.3.0, Intel Compiler version 2021.5.0, MPLAPACK 1.0.1, BNCpack 0.8, MPFR 4.1.0

OpenMP is applied for parallelization of DK method with the following compile option as `icpc -O3 -qopenmp`.

2 Brief Introduction of Current Multiple Precision Arithmetic and Performance of Current MPFR GEMM

The need for computational processing, mainly using floating-point arithmetic, will increase and not decrease in the foreseeable future, not only for scientific computing, but also for deep learning and other applications. The performance of hardware such as CPUs and GPUs is mainly improved in the following two ways:

- Parallel processing capabilities such as SIMD instructions and an increase in the number of cores in CPUs and GPUs,
- Increasing or decreasing the length of mantissa and exponent in floating-point numbers according to the necessity of user's computation, such as the introduction of half-precision and single-precision floating-point operations.

In similar ways, the whole process of computation is now accelerated by enhancing parallelization techniques. While taking advantage of these hardware performance improvement and employing parallelization techniques such as SIMD instruction, OpenMP, and MPI, mixed precision techniques combining IEEE binary32, binary64, and half precision arithmetic, which are standard in hardware, are also used. Regarding MPF operations using software libraries, which are adopted in adverse conditions where binary64 arithmetic lacks computational accuracy, the heavy MPF processing requires the active use of similar software performance improvement techniques.

Currently, the mainstream of MPF arithmetic falls into two types of implementations: a multi-component method that combines multiple binary32 and binary64, using error-free transformation (EFT) techniques to extend the mantissa length, and an integer-based many-digit method. The QD library by Bailey et al. [2] is well-known for those based on the multi-component method, and the MPFR library using the arbitrary-length natural number kernel (MPN) of GNU MP (GMP) has a significant number of users because of its superior speed and reliability. MPLAPACK by Maho Nakata is a multiple precision linear computation library based on the C++ converted from the original Fortran code of LAPACK/BLAS, which was developed from these two multiple precision floating-point arithmetic libraries, and the latest version (Version 1.0.1) as of February 2022, provides the parallelized BLAS code with OpenMP. In addition, the main driver and calculation routines of LAPACK are available in multiple precision. However, we will wait to gain all driver routines benefit from this high-performance MPBLAS. In addition, SIMD instructions, the use of CUDA, and the introduction of the Ozaki scheme have not officially improved performance as faster multiple precision ATLAS, OpenBLAS, and Intel Math Kernel.

The current driver routine Rgeev of MPLAPACK used in this paper is not parallelized. In addition, the arbitrary-precision routines have been determined to be slower than our native C implementation of the basic linear subprogram, probably because of the use of MPREAL, a wrapper C++ class library for MPFR adopted in the arbitrary-precision calculations. In fact, the computational time of the MPFR block matrix multiplication supported by our BNCmatmul library is illustrated in Fig. 1 compared to the time of the MPFR Rgemm of MPBLAS. The results of Strassen matrix multiplication are also included for comparison.

Evidently, both 212 bits (the same bits of QD precision) and 1024 bits are 2.0–2.4 times (212 bits) and 1.4–1.5 times (1024 bits) larger than MPBLAS (Rgemm) by block matrix multiplication (matmul_mpfmatrix_block) in both EPYC and Xeon environments. Thus, the smaller the number of MPF mantissa, the faster the C native MPFR direct call (BNCmatmul) is than MPLAPACK/MPBLAS (MPREAL).

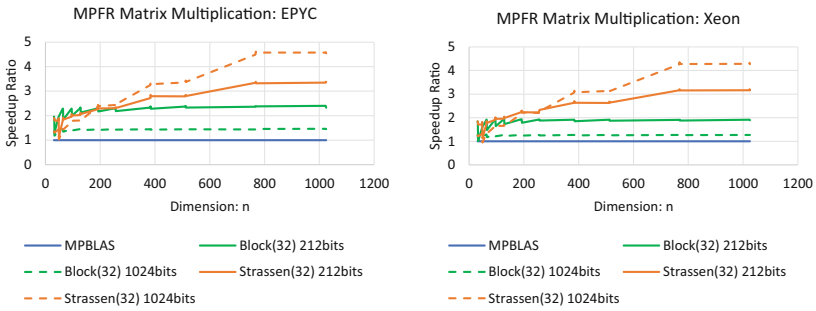

Fig. 1. Speedup ratio of MPFR matrix multiplication against MPBLAS (Rgemm)

The eigenvalue driver routine (Rgeev) for real matrices in MPLAPACK are adopted as eigensolvers, while the simultaneous iterative method for MPFR algebraic equations, prepared as a competitor, employs a direct call to MPFR for its implementation. Therefore, it is expected to perform better than the implementation using MPREAL.

3 Mixed Precision Approach for Solving Algebraic Equations

We target the following algebraic equations

$$p_n(x) = 0, \tag{1}$$

where the following real coefficient polynomial $p_n(x)$ is adopted as the left term,

$$p_n(x) = \sum_{i=0}^{n} a_i x^i \ (a_i \in \mathbb{R}, a_n \neq 0). \tag{2}$$

It is well-known that the n-th degree algebraic equation (1) definitely has n roots $\alpha_i \in \mathbb{C}$, $i = 1, 2, ..., n$ at most. In this time, suppose that we have no prerequisite for roots of the Eq. (1).

For convenience, the corresponding monic polynomial $q_n(x)$ derived from $p_n(x)$

$$q_n(x) = x^n + \sum_{i=0}^{n-1} c_i x^i \ (c_i = a_i/a_n), \tag{3}$$

is also prepared.

3.1 Eigenvalue Method for Roots of Algebraic Equation

As explained in several number of textbooks of linear algebra, it is well-known that our targeted algebraic equation (1) can be expressed by the following companion matrix C_n with the same eigenvalues as the roots of $q_n(x)$.

$$C_n = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \\ -c_0 & -c_1 & \cdots & -c_{n-2} & -c_{n-1} \end{bmatrix}. \tag{4}$$

As the polynomial (2) has only real coefficients, we can exploit xGEEV driver routines in LAPACK [8] as an eigensolver to obtain all the eigenvalues of C_n. If C_n has a negligible condition number, is diagonalizable, and does not have multiple eigenvalues, we do not require additional digits of mantissa of MPF. We call this approach the "eigenvalue method" for solving the algebraic equation (1).

3.2 Simultaneous Iteration Method for Directly Solving Algebraic Equation

The standard direct iterative method for obtaining solutions to algebraic equations of degree five or higher (1) is the Newton method and its related simultaneous methods. In recent years, several higher-order direct iterative methods have been proposed, and Petkovic has compactly summarized the results up to 2012 [11]. However, as described below, the higher the order of the direct iterative method, the more computational complexity per iteration increase, and global convergence is not guaranteed. Therefore, we employ only the second- and third-order methods [1]. The comparison of the initial guess setting methods will be discussed later.

For both second- and third-order DK methods, by expressing the approximation at k times iteration as

$$\mathbf{z}_k = [z_1^{(k)} \ z_2^{(k)} \ ... \ z_n^{(k)}]^T \in \mathbb{C}^n,$$

their iteration formulas are described with monic polynomial (3) as follows:

Second Order DK Method

$$z_i^{(k+1)} := z_i^{(k)} - \frac{q_n(z_i^{(k)})}{\prod_{j=1, j\neq i}^{n}(z_i^{(k)} - z_j^{(k)})} \tag{5}$$

Third Order DK Method

$$z_i^{(k+1)} := z_i^{(k)} - \frac{\frac{p_n(z_i^{(k)})}{p_n'(z_i^{(k)})}}{1 - \frac{p_n(z_i^{(k)})}{p_n'(z_i^{(k)})} \sum_{\substack{j=1 \\ j\neq i}}^{n} (z_i^{(k)} - z_j^{(k)})^{-1}} \tag{6}$$

To compare with our initial guess approach, we adopt Aberth's initial guesses [1] as follows:

$$z_i^{(0)} := -\frac{c_{n-1}}{n} + r \exp\left\{\left(\frac{2(i-1)\pi}{n} + \frac{3}{2n}\right)\mathrm{i}\right\}. \tag{7}$$

We adopt r in Aberth's initial guess (7) as follows:

$$r := \max_{0\leq i\leq (n-1)} |n_{\mathrm{nz}} c_i|^{1/(n-i)},$$

where $n_{\mathrm{nz}} \leq n$ is the number of non-zero coefficients in (2).

4 Examples of Ill-Conditioned Algebraic Equations

Unlike the eigenvalue problem, when solving the equation, the higher the density of roots, the larger the error in the values of the polynomial $p_n(x)$ and $q_n(x)$. Therefore, if we employ a method such as the Danilevsky method [3], where the eigenvalue problem of a matrix is transformed into a companion matrix and coefficients of the eigenequation are obtained directly, despite the good conditions for an eigenvalue problem, it becomes a bad problem to solve the algebraic equation. This is why this method via eigenequation is not recommended currently because fast eigenvalue solving methods such as the QR method, which uses a shift of the origin in the Householder transformation to obtain stable and accurate eigenvalues exist.

However, regarding bad algebraic equations that require longer MPF numbers than binary64 for the coefficients, MPF operations are essential in the process of solving, and there is a possibility that the solution of algebraic equations with less computational complexity can be performed faster than matrix eigenvalue routines. In our previous study on the Gauss-type integral formulas for the quantile calculations [7], there were several cases where the computation time was reduced using the Newton method with the approximate eigenvalues obtained by binary64 as the initial guess, rather than computing all eigenvalues using the MPF eigensolver for real symmetric matrices. As an extension of this

result, it is expected that similar mixed-precision techniques will be effective in reducing computation time for solving arbitrary real coefficient algebraic equations using a real asymmetric companion matrix to derive useful initial guesses.

Here, we consider two examples: Wilkinson's example (with all real roots) and an algebraic equation whose solution is the quantile of the Chebyshev quadrature formula (with almost all complex roots) to prove that such examples exist in the benchmark test.

Wilkinson's Example

This well-known example is explained in detail in Wilkinson's book [16].

The polynomial (2) is provided as $p_n(x) = \prod_{i=1}^{n}(x-i)$, with $\alpha_i = i$ as its roots. The absolute value of coefficients is glowing up on $O(n!)$. For $n = 20$, the coefficients are

$$\begin{aligned} a_0 &= 2432902008176640000 \\ a_1 &= -8752948036761600000 \\ &\vdots \\ a_{19} &= -210 \\ a_{20} &= 1. \end{aligned}$$

In the case of $n = 128$ adopted in our experiments, $a_0 = 3.8562\cdots \times 10^{215}$ means that coefficients at over $n = 128$ may not be treated in binary64, DD (double-double, 106 bits), and QD (quad-double, 212 bits) arithmetic. Figure 2 illustrates how the approximation of the roots with DD, QD, and MPREAL 512 bits Rgeev are distributed on Gauss plane. Although DD and QD Rgeev produce different but results located near each other on the Gauss plane, it is obvious that DD and QD Rgeev cannot provide an accurate approximation of the roots.

However, as indicated later, DD approximations are useful as initial guesses to accelerate MPF direct iterative methods.

Chebyshev Quadrature Problem

As previously described, this example is solely popular in Japan. In this subsection, we mathematically explain the Chebyshev quadrature problem [15].

The Chebyschev quadrature formula is not categorized in the Gaussian quadrature formula. When $[-1, 1]$ is provided as the integration interval, we can obtain the following discrete quadrature formula as follows:

$$\int_{-1}^{1} w(x)f(x)dx \approx \sum_{k=1}^{n} w_k(x_k)f(x_i). \tag{8}$$

In the above formula (8), we suppose that $w(x) = 1$ and $w_k(x_k) = 2/n$ is fixed. In this case, the "Chebyshev quadrature formula" is determined by choosing the appropriate abscissas.

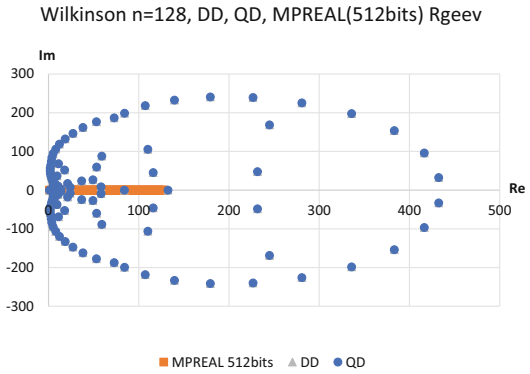

Fig. 2. Approximation of DD, QD, and MPREAL (512 bits) Rgeev (MPLAPACK) on Gauss plane: Wilkinson's example $n = 128$

The coefficients of polynomial (2) with roots as their abscissas are derived, starting with $a_n = 1$, as follows:

$$\begin{cases} a_{n-(2k-1)} := 0 \\ a_{n-2k} := -\sum_{2j+1}^{k} a_{n-2(k-j)} \end{cases}, \tag{9}$$

where $k = 1, 2, ..., \lfloor n/2 \rfloor$. The odd numbered terms are zero, and the even numbered terms are derived from (9).

The algebraic equation (1) with the above coefficients has real roots only in the case of $n = 1, 2, ..., 7$, 9, and almost all conjugate complex pairs of roots in the case of $n = 8$ or $n \geq 10$. According to Moriguchi and Iri's prediction [5], regarding $n \to \infty$, all roots are nearly distributed on the curve and expressed as the following functions of z,

$$|(z+1)^{(z+1)/2}(z-1)^{-(z-1)/2}| = 2. \tag{10}$$

The prediction has been precisely confirmed in numerical experiments through Ono's works [12,13].

In addition, we should handle the process of calculating the coefficients using the formula (9). With an increase in the order of the polynomial, the digits decrease significantly, and longer MPF operations are required to obtain accurate coefficients. According to studies of Masumoto et al., we need over 215 decimal digits to guarantee the over six decimal significant digits of a_0 in case of $n = 1024$. As illustrated in Fig. 3, DD and QD precision arithmetic are not sufficient to obtain accurate roots even in the case of $n = 256$.

5 Benchmark Tests

Our mixed precision approach for targeted algebraic equation (1) is set up as follows:

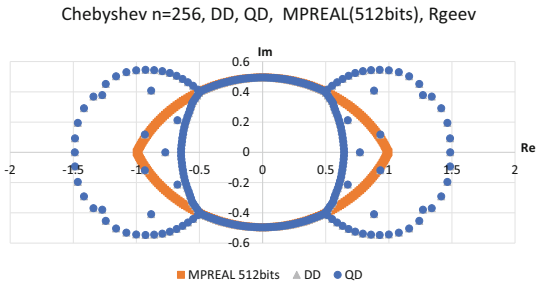

Fig. 3. The abscissas of Chebyshev quadrature problem ($n = 256$) obtained by DD, QD, and MPREAL (512 bits) Rgeev on Gauss plane

1. Calculate all coefficients a_i, $i = 0, 1, ..., n$ of the algebraic equation with L bits MPFR arithmetic.
2. Transfer MPFR a_i to the DD ones and then calculate only eigenvalues $\lambda_i^{(\mathrm{DD})} \in \mathbb{C}$ of the corresponding companion matrix C using DD Rgeev of MPLAPACK.
3. Set the above DD eigenvalues $\lambda_i^{(\mathrm{DD})}$ as initial guesses $\mathbf{z}_0 \in \mathbb{C}^n$ for second- or third- order DK methods. Converged values $\mathbf{z}_{\mathrm{end}} \in \mathbb{C}^n$ are adopted as final approximations of roots α_i.

Although the value of $|p_n(x)|$ is normally used for the stopping rule of iteration, we use the difference of $z_i^{(k)}$ and $z_i^{(k+1)}$, relative tolerance $\varepsilon_{\mathrm{rel}}$, and absolute tolerance $\varepsilon_{\mathrm{abs}}$ as follows:

$$|z_i^{(k+1)} - z_i^{(k)}| \le \varepsilon_{\mathrm{rel}} |z_i^{(k)}| + \varepsilon_{\mathrm{abs}}. \tag{11}$$

We recognize that the $(k+1)$-th iterated approximation is converged when the condition of (11) is satisfied. The relative errors as illustrated in latter figures are calculated by comparing with the eigenvalues using MPREAL 2048 bits Rgeev.

5.1 Wilkinson's Example

First, we relay the results for Wilkinson's example ($n = 128$) on EPYC and Xeon.

As shown in Fig. 4, the 512 bit MPREAL Rgeev can obtain an accurate approximation of the roots. In contrast, to obtain an approximate solution with sufficient accuracy using the DK method, the 1024 bits are necessary, and $\varepsilon_{\mathrm{rel}} := 7.5 \times 10^{-145}$ and $\varepsilon_{\mathrm{abs}} := 1.0 \times 10^{-300}$ are adopted to determine convergence. Accordingly, we can confirm that the approximations of both ways can reach the same level of relative errors as illustrated by Fig. 4.

The entire results via benchmark tests are presented in Table 1, where "DKA2" (second order DK) and "DKA3" (third order DK) indicate the computational time (in seconds) and number of iterations for each number of threads

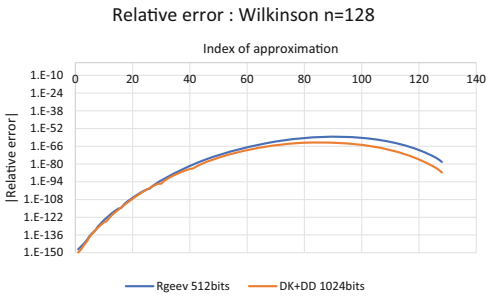

Fig. 4. Relative errors of Rgeev (512 bits) and DK-DD (1024 bits) for Wilkinson's example: $n = 128$.

when the initial guess of Aberth is employed, and "DK2+DD" and "DK3+DD" indicate the computational time and number of iterations when DD Rgeev is adopted to derive the initial guess. Currently, the Rgeev of MPLAPACK is not parallelized; hence, it cannot be accelerated with over two threads. Underlined computational time means being faster than those of MPFR Rgeev.

Table 1. Wilkinson's example:$n = 128$, Rgeev 512 bits, DK methods 1024 bits

EPYC	Rgeev (Second)		DKA2 (1024 bits)		DKA3 (1024 bits)		DK2 (1024 bits)+DD		DK3 (1024 bits)+DD	
#Thr.	DD	MPFR 512 bits	Second	#Iter.	Second	#Iter.	Second	#Iter.	Second	#Iter.
1	0.106	5.1	79.7	1374	71.1	691	30.8	538	51.5	512
2			39.8	1374	36.6	691	15.4	535	26.1	509
4			20	1374	18.7	691	8.14	562	14	512
8			10.1	1374	8.96	691	1.43	195	1.42	99
16			5.15	1374	4.55	691	2.07	557	3.32	511
24			3.88	1374	3.74	691	1.52	541	2.69	511
Xeon	Rgeev (Second)		DKA2 (1024 bits)		DKA3 (1024 bits)		DK2 (1024 bits)+DD		DK3 (1024 bits)+DD	
#Thr.	DD	MPFR 512 bits	Second	#Iter.	Second	#Iter.	Second	#Iter.	Second	#Iter.
1	0.09	3.5	58.6	1374	50.8	691	22.3	538	37.1	512
2			29.2	1374	26.2	691	11.3	535	18.8	509
4			15.3	1374	13.7	691	6.18	562	9.93	512
8			7.78	1374	6.86	691	1.1	195	0.996	99
16			4.73	1374	4.16	691	1.86	557	2.97	511
18			4.88	1374	4.37	691	1.9	557	3.05	511

We can observe the following results from Table 1.

1. When Aberth's initial values are adopted, the number of iterations for DK3 is approximately half that of DK2. However, the computation time has not decreased much.
2. When using the eigenvalues of DD Rgeev as initial values, the number of iterations is reduced compared to when using Aberth's initial values, and the

computation time is also reduced. In addition, the number of iteration varies from thread to thread, and the change in $\lambda_i^{(\mathrm{DD})}$ has a significant effect on the iterative process.

The speedup ratio by parallelization with OpenMP is illustrated in Fig. 5.

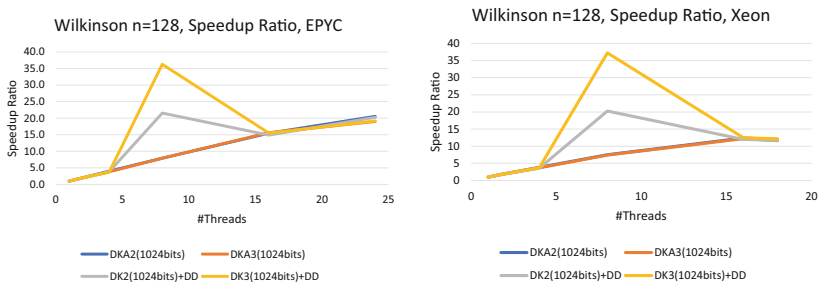

Fig. 5. Speedup ratio of Wilkinson's problem

These figures illustrate that parallelization is efficient for direct iterative methods, and that the decrease of iterative times pull up the speedup ratio at DK2+DD and DK3+DD using eight threads.

5.2 Chebyshev Quadrature Problem

Here, we explain the results of solving the Chebyshev quadrature formula quantile problem ($n = 256$ and $n = 512$).

First we present the case of $n = 256$. MPREAL (256 bits) is adopted to derive the coefficients based on the (9) formula, and the DK methods employs a 512 bits calculation and $\varepsilon_{\mathrm{rel}} := 8.6 \times 10^{-68}$ and $\varepsilon_{\mathrm{abs}} := 1.0 \times 10^{-300}$ for convergence determination. As illustrated in Fig. 6, this results in accurate approximations of roots of approximately 30 decimal digits.

Table 2 presents all the results of the benchmark tests. MPREAL (256 bits) Rgeev took 57.6 s on EPYC and 38.4 on Xeon, respectively. The DK methods employ a 512 bits MPF arithmetic, $\varepsilon_{\mathrm{rel}} := 8.6 \times 10^{-68}$ and $\varepsilon_{\mathrm{abs}} := 1.0 \times 10^{-300}$ for convergence determination.

The common numerical properties and trend of computational times is presented in Table 2. In addition, we observe that the serial DK2+DD is faster than MPREAL Rgeev.

According to the results presented in Table 2 and Fig. 7 illustrates the speedup ratio. We can confirm that the second- and third-order DK methods have approximately achieved an ideal speedup ratio. As seen in Wilkinson's example, a decrease in the number of iterations in DK methods increases the speedup ratio.

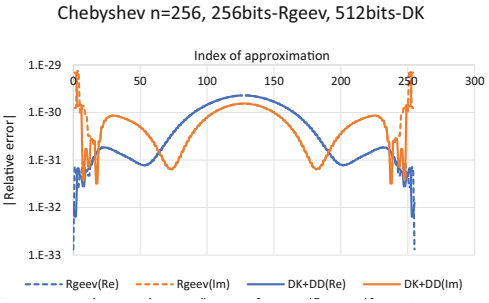

Fig. 6. Relative errors of Rgeev (256 bits) and DK-DD (512 bits) for the Chebyshev quadrature problem: $n = 256$.

Table 2. Chebyshev quadrature: $n = 256$, Rgeev 256 bits, DK methods 512 bits

EPYC	Rgeev (Second)		DKA2 (512 bits)		DKA3 (512 bits)		DK2 (512 bits)+DD		DK3 (512 bits)+DD	
#Thr.	DD	MPFR 256 bits	Second	#Iter.	Second	#Iter.	Second	#Iter.	Second	#Iter.
1	1.29	57.6	203.0	1344	183	671	47	316	70.3	264
2			100.0	1344	93.8	671	7.11	96	10	73
4			51.1	1344	46.3	671	3.62	96	5.08	71
8			25.3	1344	23.1	671	5.77	308	8.92	264
16			13	1344	11.7	671	0.918	95	1.22	71
24			8.96	1344	8.1	671	2.02	263	3.1	263
Xeon	Rgeev (Second)		DKA2 (512 bits)		DKA3 (512 bits)		DK2 (512 bits)+DD		DK3(512 bits)+DD	
#Thr.	DD	MPFR 256 bits	Second	#Iter.	Second	#Iter.	Second	#Iter.	Second	#Iter.
1	1.12	38.4	146.0	1344	127	671	32.1	316	49.2	264
2			69.0	1344	64.8	671	4.95	96	6.91	73
4			36.3	1344	34.2	671	2.61	96	3.59	71
8			18.6	1344	17.7	671	4.2	308	6.63	264
16			10.9	1344	10	671	0.779	95	1.07	71
18			10.8	1344	9.79	671	2.32	293	3.62	250

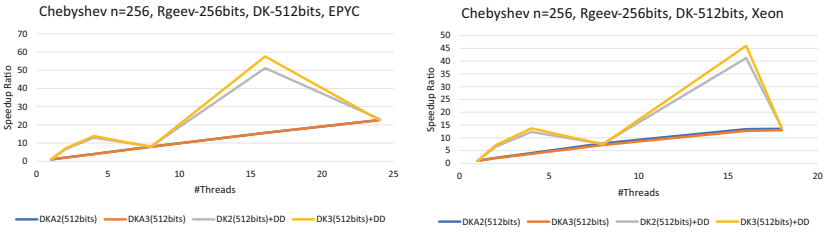

Fig. 7. Speedup Ratio of Chebyshev quadrature: $n = 256$

Second, the results of solving the Chebyshev quadrature problem in the case of $n = 512$ are indicated. MPREAL (512 bits) was adopted to derive the coefficients based on the (9) formula, and the MPREAL 512 bits Rgeev took 636.0 s and 441.0 s on EPYC and Xeon, respectively. The DK method employs a 1024 bits calculation, $\varepsilon_{\mathrm{rel}} := 7.5 \times 10^{-145}$, and $\varepsilon_{\mathrm{abs}} := 1.0 \times 10^{-300}$ for the convergence decision. This results in correct approximate solutions from 58 to 62 decimal digits, as illustrated in Fig. 8.

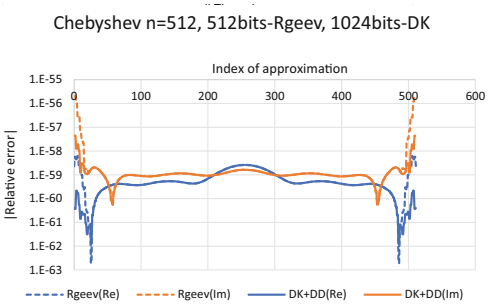

Fig. 8. Relative errors of Rgeev (512 bits) and DK-DD (1024 bits) for the Chebyshev quadrature problem: $n = 512$.

All results obtained through the benchmark test are presented in Table 3.

Table 3. Chebyshev quadrature: $n = 512$, Rgeev 512 bits, DK methods 1024 bits

EPYC	Rgeev (Second)		DKA2 (1024 bits)		DKA3 (1024 bits)		DK2 (1024 bits)+DD		DK3 (1024 bits)+DD	
#Thr.	DD	MPFR 512 bits	Second	#Iter.	Second	#Iter.	Second	#Iter.	Second	#Iter.
1	9.84	636.0	2850.0	3032	2460	1510	500	552	577	357
2			1370.0	3032	1210	1510	90.9	200	67.8	82
4			689	3032	614	1510	46.1	202	34.5	83
8			344	3032	307	1510	62.8	550	77.3	363
16			173	3032	174	1510	11.8	205	8.59	84
24			120	3032	106	1510	8.47	213	6.57	83
Xeon	Rgeev (Second)		DKA2 (1024 bits)		DKA3 (1024 bits)		DK2 (1024 bits)+DD		DK3 (1024 bits)+DD	
#Thr.	DD	MPFR 512 bits	Second	#Iter.	Second	#Iter.	Second	#Iter.	Second	#Iter.
1	9.25	441.0	2030.0	3032	1740	1510	362	552	407	357
2			1020.0	3032	937	1510	67.3	200	48.2	82
4			528	3032	471	1510	35.3	202	25.5	83
8			267	3032	238	1510	48.5	550	56.5	363
16			159	3032	142	1510	10.8	205	7.85	84
18			151	3032	132	1510	26.9	539	31.7	365

We confirm that more cases of DK2+DD and DK3+DD are faster than MPFR Rgeev.

The speedup ratio of parallelized DK methods is illustrated in Fig. 9.

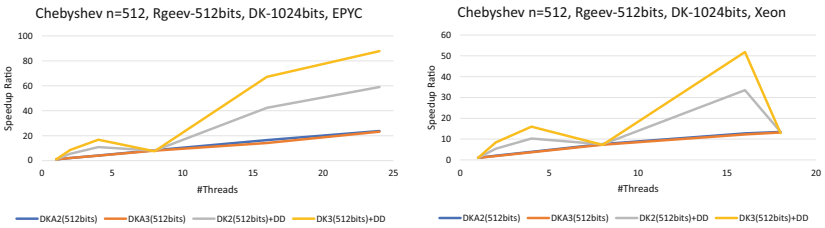

Fig. 9. Speedup Ratio of Chebyshev quadrature: $n = 512$

These two examples indicate that even if the low precision eigensolver cannot obtain good approximations, there are examples where speedup can be easily achieved using low-precision results as an initial guess for direct iterative methods with higher precision. Of course, it is not useful for all ill-conditioned algebraic equations, but the easy-to-implement and highly parallelizable direct iterative method may have a reasonably wide range of applications.

6 Conclusion and Future Works

For two types of ill-conditioned algebraic equations, initial guesses were obtained using a low-precision eigensolver, and it was indicated that a highly accurate and efficient direct iterative method can be accelerated by parallel computing. Considering the stability of the iterative computing process, it is possible to speed up the eigensolver, for example, by using the approximate eigenvalues from low-precision computation as the origin shift in high-precision computation, but the direct iterative method for algebraic equations with its computational complexity, parallelism, and ease of implementation, is also useful in long precision computing environments. In several cases, it can be stated that our mixed precision approach is useful in solving high degree algebraic equations with long precision floating-point arithmetic.

For future studies, we will implement and confirm the effectiveness of our approach as follows :

1. Application to higher degree ill-conditioned algebraic equations,
2. Implementation of other third-order (or above) direct iteration methods and comparison among them,
3. Application to general linear equations with Chebyshev Proxy Rootfinder.

In addition, we accelerate the above applications using multi-component MPF arithmetic.

Acknowledgment. This study was supported by JSPS KAKENHI, Grant Number JP20K11843, and Shizuoka Institute of Science and Technology. We acknowledge all the organizations that are continuously encouraging our study. We also appreciate the unknown referees who provided concrete and useful suggestions for revising our paper.

References

1. Aberth, O.: Iteration methods for finding all zeros of a polynomial simultaneously. Math. Comp. **27**, 339–344 (1973)
2. Bailey, D.: QD. https://www.davidhbailey.com/dhbsoftware/
3. Faddeev, D.K., Faddeeva, V.N.: An Introduction to Numerical Linear Algebra. Dover (1959)
4. Golub, G.H., Welsch, J.H.: Calculation of Gauss quadrature rules. Math. Comput. **23**, 221–230 (1969)
5. Iri, M., Yamashita, H., Terano, T., Ono, H.: An algebraic-equation solver with global convergence property. RIMS Kokuroku **339**, 43–69 (1978)
6. Kouya, T.: BNCpack. https://na-inet.jp/na/bnc/
7. Kouya, T.: Practical proposition of empirical error estimation and its application to calculation of absissas of Gauss-type integration. Trans. IPSJ **48**(SIG18(ACS20)), 1–11 (2007). (in Japanese)
8. LAPACK. http://www.netlib.org/lapack/
9. Masumoto, H., Fujino, S., Ono, H., Kojima, A.: Parallelisim of multi-precision arithmetic for computation of coefficients of algebraic equation of 20480 degrees. Trans. IPSJ **40**(12), 4159–4168 (1999). (in Japanese)
10. MPLAPACK/MPBLAS: Multiple precision arithmetic LAPACK and BLAS. https://github.com/nakatamaho/mplapack
11. Petkovic, M.S., Neta, B., Petkovic, L.D., Dzunic, J.: Multipoint Methods for Solving Nonlinear Equations. Elsevier (2013)
12. Ono, H.: On numerical computation of a high degree polynomial equation by the methods of Durand - Kerner and Aberth. Trans. IPSJ **20**(5), 399–404 (1979). (in Japanese)
13. Ono, H.: On numerical computation of a high degree polynomial equation using the method of Durand, Kerner and Aberth. Trans. IPSJ **22**(2), 165–168 (1981). (in Japanese)
14. The MPFR library. https://www.mpfr.org/
15. Weisstein, E.W.: Chebyshev Quadrature. MathWorld-A Wolfram Web Resource. https://mathworld.wolfram.com/ChebyshevQuadrature.html
16. Wilkinson, J.H.: Rounding Errors in Algebraic Process (Reprint edition). Dover (1994)
17. Yamashita, S.I.: Computation of the abscissas and weight coefficients for Gaussian quadrature formulae. IPSJ **5**, 206–215 (1964). (in Japanese)
18. Yamashita, S.I., Satake, S.: Computation of the abscissas and weight coefficients for the Hermite-Gaussian quadrature formulae. IPSJ **5**, 266–270 (1965). (in Japanese)
19. Yamashita, S.I., Satake, S.: Computation of the abscissas and weight coefficients for the Laguerre-Gaussian quadrature formulae. IPSJ **4**, 216–220 (1965). (in Japanese)