Electronic Preprint for Journal of Information Processing Vol.30

Regular Paper

Numerical Investigation into the Mixed Precision GMRES(*m*) Method Using FP64 and FP32

Yingqi Zhao[1,a)] Takeshi Fukaya[2,3,b)] Linjie Zhang[4,c)] Takeshi Iwashita[2,d)]

Received: December 17, 2021, Accepted: March 22, 2022

Abstract: In this paper, a mixed precision variant of the GMRES(*m*) method using FP64 and FP32 is investigated. Through numerical experiments for various test matrices and different settings of the restart frequency *m*, its numerical behavior is examined in detail and compared with that of the conventional GMRES(*m*) method using only FP64. From the obtained numerical results, it is confirmed that if a problem is solved by the conventional GMRES(*m*) method, basically the mixed precision GMRES(*m*) can also solve it. In addition, in the case of using small *m*, the number of the total iterations is almost equivalent between the two methods. However, when *m* grows, a different tendency is observed; the number of the iterations decreases in the conventional methods but increases in the mixed precision method. These results and observations provide new insights of the mixed precision GMRES(*m*) method, which will be helpful for the efficient use in applications and the further improvement of the method.

Keywords: sparse linear solver, iterative refinement, low precision computing, mixed precision algorithm

1. Introduction

Numerical methods for solving problems in linear algebra, e.g., system of linear equations, eigenvalue problem, singular value decomposition etc., have various applications in engineering and information science. Among them, linear solver, which solves a system of linear equations, is one of vital building blocks in scientific computations, and its computational time is usually dominant. Thus, more and more efficient linear solvers have been strongly required for accelerating application programs.

Traditionally, double-precision floating-point number (also called FP64 or float64) has been standard in most scientific computations, and FLOPS (Floating-point Operations Per Second), more precisely, FLOPS for FP64 operation, has been used as a metric for the performance of a CPU (or computer system). However, it is getting difficult to improve FP64 FLOPS of a single processor due to the limitation of power budget. In addition, the demand in market has also changed; some new applications, e.g., machine learning applications, do not strictly require FP64 and accept low precision computing that uses single-precision floating-point number (FP32) or half-precision floating-point number (FP16). These situations have triggered a paradigm shift in computer architectures; exploiting the high ability of low precision computing will be crucial, and some with specified unit for low precision computing (e.g., NVIDIA GPU with Tensor Cores) are actually now available.

In the case of using hardware equipped with specified unit for low precision computing, it is clear that computations mainly based on FP32 or FP16 have a great advantage compared with those based on FP64. Even on a standard CPU platform, low precision (currently, FP32) computing has two benefits. The first one is the reduction of the data transfer cost among memory hierarchy. When an application is memory-bound, this advantage has a significant impact on the performance. In addition, less memory footprint allows better use of the cache memory, which further improves the performance. The second one is the increase of the performance of the SIMD operation; the number of elements performed by single instruction increases. In a computation-bound application, this advantage can improve the performance.

Considering the fact that numerical methods in linear algebra have been widely used in many applications, one of promising approaches for exploiting the advantages of low precision computing in applications is developing a new method that can exploit low precision computing and provide computed results with the same accuracy as by traditional methods using only FP64. It will be easy to replace a current method with such a new method in application programs, and the performance benefit will be easily obtained.

The key idea in developing such a method is mixed precision computing, in which both the standard precision and low precision computing are combined; in the situation considered in this paper, FP64 and FP32 are combined. In this paper, the target problem is a system of linear equations whose coefficient matrix is large and sparse, and a mixed precision variant (using FP64 and FP32) of the GMRES(*m*) method, which is one of famous Krylov

1 Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Hokkaido 060–0814, Japan
2 Information Initiative Center, Hokkaido University, Sapporo, Hokkaido 060–0811, Japan
3 JST Presto, Chiyoda, Tokyo 102–0076, Japan
4 School of Mathematical Sciences, Ocean University of China, Qingdao, China
a) yqzhao2016@hotmail.com
b) fukaya@iic.hokudai.ac.jp
c) zhanglinjie@ouc.edu.cn
d) iwashita@iic.hokudai.ac.jp

subspace methods, is discussed.

In several studies [3], [4], [25], [26], [34], the effectiveness of applying mixed precision computing to the GMRES(m) method has already been presented; for some test problems, it is shown that a mixed precision GMRES(m) solver using FP64 and FP32 outperforms the traditional GMRES(m) solver using only FP64 in terms of the execution time in CPU or GPU environments. However, the numerical behavior of the mixed precision GMRES(m) method has not been sufficiently clarified yet.

This study aims for investigating the numerical behavior of the mixed precision GMRES(m) method using FP64 and FP32 in detail through numerical experiments. For various test matrices, with different parameter settings (e.g., the restart frequency m), the behavior of the mixed precision GMRES(m) is investigated and compared with that of the traditional GMRES(m) using only FP64. In addition, typical results are analyzed in detail with the residual norm histories. Presented numerical results and observations will be helpful for the efficient use of the method in applications and the further improvement of the method.

The rest of the paper is organized as follows: in Section 2, related work is summarized. In Section 3, the GMRES(m) method and the derivation of its mixed precision variant are reviewed. In Section 4, the mixed precision GMRES(m) method using FP64 and FP32 is theoretical analyzed. In Section 5, the results of the numerical experiments are presented. Finally, in Section 6, the conclusion remarks are given.

2. Related Work

There are many studies on numerical linear algebra methods that utilize mixed precision computing. The paper by Abdelfattah et al. [1] provides a good survey including studies regarding the recent hardware trend.

In numerical linear algebra, mixed precision computing has been used in the form of iterative refinement. As a technique to improve the accuracy of the computed solution of a linear system (mainly in an ill-conditioned case), mixed precision methods have been widely known [14], [17], in which higher precision computing than the standard precision (usually FP64) is partially used. It is worth noting that also for other problems than linear systems, similar methods can be found: for example, symmetric eigenvalue problems [29], [30] and inverse LU and QR factorizations [28].

For the purpose of accelerating the solution process of linear systems, mixed precision methods using lower precision computing than the standard precision have been studied more actively as the gap between the performance of the standard and lower precision computing becomes larger. In the late 2,000 s, GPGPU became widespread in scientific computations, and numerical methods that can utilize FP32 attracted much attention; the paper by Baboulin et al. [5] provides a summary. For dense linear systems, the effectiveness of the iterative refinement based method using the LU factorization computed in FP32 was presented [9], [24]. For sparse linear systems, the performance of the iterative refinement using the sparse LU factorization in FP32 was shown [8]. Approaches of using preconditioners performed in FP32 for Krylov subspace methods were also reported [8], [20], [33].

In recent years, for linear systems, mixed precision methods based on iterative refinement have been more actively studied. Carson and Higham presented a new rounding error analysis and developed a new method so-called GMRES-IR based on their analysis [10]. They also presented the GMRES-IR method using three precisions, e.g., FP64, FP32, and FP16, with its theoretical analysis [11], which was extended to least square problems [12]. Higham et al. improved the GMRES-IR method for general linear systems [19] and for symmetric positive definite problems [18], where the positive definite property should be considered in low precision Cholesky factorization. Haidar et al. reported the early performance results using FP16 provided by Tensor Cores on NVIDIA GPU [16], and Blanchard et al. presented error analysis of the mixed precision block FMM operation by Tensor Cores [7], which is a building block in studies on mixed precision methods using Tensor Cores. Based on the above studies, a new benchmark test, namely HPL-AI [*1], was designed for measuring the low precision computing performance of supercomputer systems; for example, it was already performed on the supercomputer Fugaku [23]. It is worth noting that the GMRES-IR method employs the full LU factorization computed in low precision as a preconditioner and is assumed to converge within a few iterations. Thus, although it includes the term "GMRES", it is essentially different from the GMRES method used in solving sparse linear systems.

In the context of accelerating the Krylov subspace methods for solving sparse linear systems, a mixed precision variant of the GMRES(m) method using FP32 and FP64 was presented by Turner and Walker [34]. Anzt et al. presented the performance results on a GPU platform [3], [4]. Recently, Lindquist et al. have provided detailed performance results of both the unpreconditioned and preconditioned mixed precision GMRES(m) methods on a modern CPU platform [25], [26]. The evaluation on a modern GPU platform has been also reported by Loe et al. [27]. Algorithms presented in these studies are essentially equivalent; they are all based on the structure of iterative refinement in the GMRES(m) method, which is reviewed in Section 3. Various examples of accelerating the GMRES(m) method by utilizing FP32 have been shown in these studies, which confirm the usefulness of the mixed precision GMRES(m) method. However, these papers presented almost only successful results, and the whole picture of the method seems to be still not clear, which is the motivation of this paper.

Also for other iterative methods for sparse linear systems, e.g., the CG method, the use of low precision computing was studied [2], [15]. In these studies, approaches based on iterative refinement were considered. However, different from the situation in the GMRES(m) method, target methods such as the CG method are generally performed without restart, which means that the methods do not have the structure of interactive refinement. Thus, the effectiveness of the developed mixed precision methods is limited compared with the original methods.

[*1] https://hpl-ai.org/

3. Review of the GMRES(m) and Mixed Precision GMRES(m) Methods

In this section, a brief introduction of the GMRES(m) method is first given. Then, the mixed precision GMRES(m) is shown together with the idea behind its derivation.

3.1 The GMRES(m) Method

The GMRES (Generalized Minimal RESidual) method [32], proposed in 1986 by Saad and Schultz [31], is one of standard iterative methods for solving a linear system with a large, sparse and general (non symmetric) coefficient matrix. In this paper, we consider solving a system of linear equations

$$Ax = b, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is the coefficient matrix, $x \in \mathbb{R}^n$ is the solution vector and $b \in \mathbb{R}^n$ is the right-hand side vector.

Let x_k be an approximate solution of the linear system (1) obtained at the k-th iteration of the GMRES method. Given an initial guess x_0, the GMRES method finds an approximate solution x_k from the affine space $x_0 + \mathscr{K}_k(A, r_0)$, where $r_0 = b - Ax_0$ and $\mathscr{K}_k(A, r_0) = \mathrm{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$, which is called the Krylov subspace.

In a typical algorithm of the GMRES method, an orthonormal matrix $V_k = \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix} \in \mathbb{R}^{n\times k}$, whose column vectors form an orthogonal basis of $\mathscr{K}_k(A, r_0)$, and an upper Hessenberg matrix

$$\tilde{H}_k = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} \\ & h_{3,2} & \cdots & h_{3,k} \\ & & \ddots & \vdots \\ & & & h_{k+1,k} \end{pmatrix} \in \mathbb{R}^{(k+1)\times k} \tag{2}$$

are computed by the Arnoldi process involving sparse matrix vector multiplication (SpMV) and the modified Gram-Schmidt orthogonalization (MGS). The approximate solution x_k is obtained as

$$x_k = x_0 + V_k y_k, \tag{3}$$

where y_k is determined so as to minimize the 2-norm of the residual vector $r_k = b - Ax_k$:

$$\begin{aligned} y_k &= \arg\min_{y\in\mathbb{R}^k} \|r_k\|_2 \\ &= \arg\min_{y\in\mathbb{R}^k} \|b - A(x_0 + V_k y)\|_2 \\ &= \arg\min_{y\in\mathbb{R}^k} \|r_0 - AV_k y\|_2. \end{aligned} \tag{4}$$

Using the relation

$$AV_k = V_{k+1}\tilde{H}_k, \tag{5}$$

the minimization problem (4) can be transformed as follows:

$$\begin{aligned} \min_{y\in\mathbb{R}^k} \|r_0 - AV_k y\|_2 &\Leftrightarrow \min_{y\in\mathbb{R}^k} \|r_0 - V_{k+1}\tilde{H}_k y\|_2 \\ &\Leftrightarrow \min_{y\in\mathbb{R}^k} \|V_{k+1}^\top r_0 - \tilde{H}_k y\|_2 \\ &\Leftrightarrow \min_{y\in\mathbb{R}^k} \|\beta u - \tilde{H}_k y\|_2, \end{aligned} \tag{6}$$

Algorithm 1 GMRES(m)

Input: x_0: initial guess, ϵ: convergence criterion, A; coefficient matrix, b: right-hand side vector, m: restart frequency, maximum number of total inner iterations

1: **repeat**
2: $r_0 = b - Ax_0$
3: $\beta = \|r_0\|_2$
4: **if** $\beta/\|b\|_2 \le \epsilon$ **then return** x_0
5: $v_1 = r_0/\beta$
6: Compute V_{m+1} and $\tilde{H}_m$ by the m-step Arnoldi process with A and v_0.
7: Compute y_m from β and $\tilde{H}_m$.
8: $x_m = x_0 + V_m y_m$
9: $x_0 = x_m$
10: **until** attain the maximum number of total inner iterations

Output: x_0

where $\beta = \|r_0\|_2$ and $u = [1, 0, \ldots, 0]^\top \in \mathbb{R}^{k+1}$. In the standard algorithm, the minimization problem (6) is solved by the QR factorization based on the Givens rotation.

The GMRES method has a good convergence property, however, both the computational cost per iteration and the memory footprint grow as the number of iteration (i.e., k) increases. Due to this issue, naively using the GMRES method for solving a large problem is often difficult in a practical situation, and the restart variant of the GMRES method [32], which is known as the GMRES(m) method, is widely used.

In the GMRES(m) method, the parameter m is the restart frequency, and the following two steps are repeated:

- **Step 1:** Solve $Ax = b$ by the m-iteration GMRES with the initial guess x_0 and obtain the approximate solution x_m.
- **Step 2:** Update the initial guess: $x_0 = x_m$.

The outline of the GMRES(m) method is shown in Algorithm 1. Since the m-step Arnoldi process is an iterative computation, Algorithm 1 has a nested loop structure; hereafter, we use the term *inner iteration* when referring to the iteration of the m-step Arnoldi process in the GMRES(m) method. It is known that the convergence rate of the GMRES(m) tends to be better, i.e., the number of total inner iterations tends to decrease, as a larger m is chosen. However, a larger m does not always lead to the reduction in computational time because m also affects the time per iteration of the m-step Arnoldi process.

3.2 The Mixed Precision GMRES(m) Method

The mixed precision variant of the GMRES(m) method studied in this paper has the mathematical background related to the iterative refinement scheme for the solution of a linear system [14], [17]. The iterative refinement scheme consists of the following three steps: let $\tilde{x}$ be the current approximate solution for $Ax = b$, then

- **Step 1:** Compute the residual $r = b - A\tilde{x}$.
- **Step 2:** Approximately solve the error equation $Ae = r$ and obtain an approximate solution $\tilde{e}$.
- **Step 3:** Update the solution $\tilde{x} = \tilde{x} + \tilde{e}$.

A mixed precision variant of the above iterative refinement scheme is well-known, in which $\tilde{x}$ is stored in *high precision*, r is computed by *high precision*, but $Ae = r$ is solved by *low precision*.

Algorithm 2 Mixed Precision GMRES(m)

Input: x_0: initial guess, ϵ: convergence criterion, A; coefficient matrix, b: right-hand side vector, m: restart frequency, maximum number of total inner iterations

1: $A^{(\mathrm{L})} = \mathrm{Low}(A)$
2: **repeat**
3: $r_0 = b - Ax_0$
4: $\beta = \|r_0\|_2$
5: **if** $\beta/\|b\|_2 \leq \epsilon$ **then return** x_0
6: $v_1 = r_0/\beta$
7: $\beta^{(\mathrm{L})} = \mathrm{Low}(\beta)$
8: $v_0^{(\mathrm{L})} = \mathrm{Low}(v_0)$
9: Compute $V_{m+1}^{(\mathrm{L})}$ and $\tilde{H}_m^{(\mathrm{L})}$ by the m-step Arnoldi process with $A^{(\mathrm{L})}$ and $v_0^{(\mathrm{L})}$. ▷ using *low precision* arithmetic
10: Compute $y_m^{(\mathrm{L})}$ from $\beta^{(\mathrm{L})}$ and $\tilde{H}_m^{(\mathrm{L})}$. ▷ using *low precision* arithmetic
11: $z_m^{(\mathrm{L})} = V_m^{(\mathrm{L})} y_m^{(\mathrm{L})}$ ▷ using *low precision* arithmetic
12: $z_m = \mathrm{Std}(z_m^{(\mathrm{L})})$
13: $x_m = x_0 + z_m$
14: $x_0 = x_m$
15: **until** attain the maximum number of total inner iterations

Output: x_0

It was pointed out by Imakura et al. [21] that the following computation in the GMRES(m) method:

- **Step 1:** Solve $Ax = b$ by the m-iteration GMRES with the initial guess x_0 and obtain the approximate solution x_m.
- **Step 2:** Update the initial guess: $x_0 = x_m$.

is mathematically equivalent to the following computation:

- **Step 1:** Solve $Ae = r$ by the m-iteration GMRES with the initial guess $e_0 = 0$ and obtain the approximate solution e_m, where $r = b - Ax_0$.
- **Step 2:** Update the initial guess: $x_0 = x_0 + e_m$.

This relation means that the GMRES(m) method can be interpreted as the iterative refinement that uses the m-iteration GMRES method for solving error equations. Then, a mixed precision variant of the iterative refinement with the m-iteration GMRES method is easily obtained in the above manner, which also can be interpreted as a reasonable mixed precision variant of the GMRES(m) method [*2].

The outline of the derived algorithm of the mixed precision GMRES(m) method is shown in Algorithm 2. In this algorithm, variables with the suffix "(L)" are stored in *low precision*, and Low($\cdot$) and Std($\cdot$) mean the operation of converting a variable from *standard precision* to *low precision* and from *low precision* to *standard precision*, respectively. It is worth noting that the algorithm presented here is essentially not new, and similar algorithms can be found in the related studies [3], [4], [25], [26], [34].

4. Theoretical Analysis of the Mixed Precision GMRES(m) Method Using FP64 and FP32

In this section, the mixed precision GMRES(m) method using FP64 and FP32 is theoretically analyzed, mainly compared with the traditional GMRES(m) method using only FP64. Hereafter, the former is denoted as "MP-GMRES(m)", and the latter as "FP64-GMRES(m)".

[*2] A mixed precision variant of the iterative refinement that uses the Lanczos/Bi-Lanczos type method (e.g., CG or Bi-CG) can be naturally obtained. However, unlike the case of GMRES(m), it is difficult to regard the resulting method as a reasonable mixed precision variant of the original method; there is no mathematical relation to the iterative refinement.

4.1 Details of the Algorithm of MP-GMRES(m)

The outline of the mixed precision GMRES(m) method is already shown in Algorithm 2, however its details are not clear, and we here present them. Algorithm 3 presents the details of MP-GMRES(m) investigated in this research, in which a variable with the suffix "(FP32)" is stored in FP32, and a line with the comment "by FP32" is computed by the FP32 arithmetic. k_{total} counts the total number of inner iterations. It is worth noting that Algorithm 3 is equivalent to the algorithm presented in the previous studies [25], [26].

4.2 Observations on the Mathematical Behavior

The following observations are obtained on the mathematical behavior of MP-GMRES(m) shown in Algorithm 3.

- The approximate solution x_0 is stored in FP64, and its accuracy is verified using A and b, which are also stored in FP64 (Lines 7–9). This is equivalent to FP64-GMRES(m), and the returned solution is as accurate as that by FP64-GMRES(m) in terms of the residual norm if the convergence criterion is satisfied.
- In the computation of the m-step GMRES process (Lines 11–33), all variables are stored in FP32, and all computations are performed in FP32. Thus, the refinement ratio of the approximate solution by each m-step GMRES process is expected to be less than that in FP64-GMRES(m), and it means that the number of the outer iterations (i.e., the loop from Line 6 to Line 37) will likely increase. However, if the approximate solution is refined even a little in every m-step GMRES process, which means

$$\|b - Ax_0\|_2 > \|b - Ax_k\|_2 \tag{7}$$

  at Line 35, MP-GMRES(m) can attain the convergence.
- The use of FP32 in the computation of the m-step GMRES process of MP-GMRES(m) impacts the following two mathematical aspects (here, rounding error is ignored):
  - The Krylov subspace $\mathcal{K}_k(A^{(\mathrm{FP32})}, v_1^{(\mathrm{FP32})})$ is generated in MP-GMRES(m) (Lines 16–22), which is different from that in FP64-GMRERS(m) (i.e., $\mathcal{K}_k(A, v_1)$). However, it is not clear how this difference impacts the convergence behavior of the method.
  - In the context of the iterative refinement, each m-step GMRES process of MP-GMRES(m) solves

$$A^{(\mathrm{FP32})} e = r_0^{(\mathrm{FP32})} (= \beta^{(\mathrm{FP32})} v_1^{(\mathrm{FP32})}), \tag{8}$$

    which is a perturbed linear system from

$$Ae = r_0 (= \beta v_1). \tag{9}$$

    Thus, finding a better solution for (8), which generally corresponds to employing a larger m, may negatively impact on solving the original linear system (1). This indicates that investigating into the relationship between m and the convergence behavior of MP-GMRES(m) is an important

Algorithm 3 MP-GMRES(m) investigated in this research

Input: x_0: initial guess, ϵ: convergence criterion, A; coefficient matrix, b: right-hand side vector, m: restart frequency, $k_{\max}$: maximum number of total inner iterations

1: $A^{(\mathrm{FP32})} = \mathrm{ToFP32}(A)$
2: $\zeta = \|b\|_2$
3: $\zeta^{(\mathrm{FP32})} = \mathrm{ToFP32}(\zeta)$
4: $\epsilon^{(\mathrm{FP32})} = \mathrm{ToFP32}(\epsilon)$
5: $k_{\mathrm{total}} = 0$
6: **repeat**
7: $r_0 = b - Ax_0$
8: $\beta = \|r_0\|_2$
9: **if** $\beta/\zeta \le \epsilon$ **then return** x_0
10: $v_1 = r_0/\beta$
11: $\beta^{(\mathrm{FP32})} = \mathrm{ToFP32}(\beta)$
12: $v_1^{(\mathrm{FP32})} = \mathrm{ToFP32}(v_1)$
13: $u^{(\mathrm{FP32})} = [u_1^{(\mathrm{FP32})}, \ldots, u_{k+1}^{(\mathrm{FP32})}]^\top = [\beta^{(\mathrm{FP32})}, 0, \ldots, 0]^\top$
14: **for** $k = 1, 2, \ldots, m$ **do**
15: $k_{\mathrm{total}} = k_{\mathrm{total}} + 1$
16: $w^{(\mathrm{FP32})} = A^{(\mathrm{FP32})} v_k^{(\mathrm{FP32})}$ ▷ by FP32
17: **for** $j = 1, 2, \ldots, k$ **do**
18: $h_{j,k}^{(\mathrm{FP32})} = v_j^{(\mathrm{FP32})\top} w^{(\mathrm{FP32})}$ ▷ by FP32
19: $w^{(\mathrm{FP32})} = w^{(\mathrm{FP32})} - h_{j,k}^{(\mathrm{FP32})} v_j^{(\mathrm{FP32})}$ ▷ by FP32
20: **end for**
21: $h_{k+1,k}^{(\mathrm{FP32})} = \|w^{(\mathrm{FP32})}\|_2$ ▷ by FP32
22: $v_{k+1}^{(\mathrm{FP32})} = w^{(\mathrm{FP32})}/h_{k+1,k}^{(\mathrm{FP32})}$ ▷ by FP32
23: **for** $j = 1, 2, \ldots, k-1$ **do**
24: $\begin{pmatrix} h_{j,k}^{(\mathrm{FP32})} \\ h_{j+1,k}^{(\mathrm{FP32})} \end{pmatrix} = \begin{pmatrix} c_j^{(\mathrm{FP32})} & s_j^{(\mathrm{FP32})} \\ -s_j^{(\mathrm{FP32})} & c_j^{(\mathrm{FP32})} \end{pmatrix} \begin{pmatrix} h_{j,k}^{(\mathrm{FP32})} \\ h_{j+1,k}^{(\mathrm{FP32})} \end{pmatrix}$ ▷ by FP32
25: **end for**
26: $c_k^{(\mathrm{FP32})} = h_{k,k}^{(\mathrm{FP32})} / \sqrt{h_{k,k}^{(\mathrm{FP32})^2} + h_{k+1,k}^{(\mathrm{FP32})^2}}$ ▷ by FP32
27: $s_k^{(\mathrm{FP32})} = h_{k+1,k}^{(\mathrm{FP32})} / \sqrt{h_{k,k}^{(\mathrm{FP32})^2} + h_{k+1,k}^{(\mathrm{FP32})^2}}$ ▷ by FP32
28: $\begin{pmatrix} u_k^{(\mathrm{FP32})} \\ u_{k+1}^{(\mathrm{FP32})} \end{pmatrix} = \begin{pmatrix} c_k^{(\mathrm{FP32})} & s_k^{(\mathrm{FP32})} \\ -s_k^{(\mathrm{FP32})} & c_k^{(\mathrm{FP32})} \end{pmatrix} \begin{pmatrix} u_k^{(\mathrm{FP32})} \\ u_{k+1}^{(\mathrm{FP32})} \end{pmatrix}$ ▷ by FP32
29: $\begin{pmatrix} h_{k,k}^{(\mathrm{FP32})} \\ h_{k+1,k}^{(\mathrm{FP32})} \end{pmatrix} = \begin{pmatrix} c_k^{(\mathrm{FP32})} & s_k^{(\mathrm{FP32})} \\ -s_k^{(\mathrm{FP32})} & c_k^{(\mathrm{FP32})} \end{pmatrix} \begin{pmatrix} h_{k,k}^{(\mathrm{FP32})} \\ h_{k+1,k}^{(\mathrm{FP32})} \end{pmatrix}$ ▷ by FP32
30: **if** $|u_{k+1}^{(\mathrm{FP32})}|/\zeta^{(\mathrm{FP32})} \le \epsilon^{(\mathrm{FP32})}$ **then goto** 32 ▷ by FP32
31: **end for**
32: $y_k^{(\mathrm{FP32})} = \begin{pmatrix} h_{1,1}^{(\mathrm{FP32})} & \cdots & h_{1,k}^{(\mathrm{FP32})} \\ & \ddots & \vdots \\ & & h_{k,k}^{(\mathrm{FP32})} \end{pmatrix}^{-1} \begin{pmatrix} u_1^{(\mathrm{FP32})} \\ \vdots \\ u_k^{(\mathrm{FP32})} \end{pmatrix}$ ▷ by FP32
33: $z_k^{(\mathrm{FP32})} = \begin{bmatrix} v_1^{(\mathrm{FP32})} & \cdots & v_k^{(\mathrm{FP32})} \end{bmatrix} y_k^{(\mathrm{FP32})}$ ▷ by FP32
34: $z_k = \mathrm{ToFP64}(z_k^{(\mathrm{FP32})})$
35: $x_k = x_0 + z_k$
36: $x_0 = x_k$
37: **until** $k_{\mathrm{total}} > k_{\max}$

Output: x_0

task in this study.

- In Algorithm 3, the relative residual 2-norm is checked in each inner iteration (Line 30), and the m-step GMRES process breaks if the criterion is satisfied. However this check is implicitly performed for the perturbed system (8) with FP32 variables by FP32 arithmetic. Therefore, a solution x_k that passes the check at Line 30 may fail to pass the check at Line 9, which means that more m-step GMRES processes are required to satisfy the convergence criterion. Since the computation of the check at Line 9 needs SpMV with FP64 variables by FP64 arithmetic, this wrong judgement may increase the execution time, and there is a room to improve how to break the m-step GMRES process.

Table 1 Comparison of memory footprint (in bytes) between FP64-GMRES(m) and MP-GMRES(m).

			FP64	MP
A	`val`	(FP64)	$8N_{\mathrm{nz}}$	$8N_{\mathrm{nz}}$
	`col_ind`	(INT32)	$4N_{\mathrm{nz}}$	$4N_{\mathrm{nz}}$
	`row_ptr`	(INT32)	$4n$	$4n$
	`val`	(FP32)	–	$4N_{\mathrm{nz}}$
V		(FP64)	$8(m+1)n$	–
		(FP32)	–	$4(m+1)n$
Total			$12N_{\mathrm{nz}}$ $+(8m+12)n$	$16N_{\mathrm{nz}}$ $+(4m+8)n$

4.3 Remarks on the Implementation on a Standard CPU Platform

In a computational environment based on standard CPU(s), both FP64 and FP32 are available (e.g., double and float in C language), and converting a variable between these two data types is also easy (e.g., type casting in C). Thus, a main concern when implementing MP-GMRES(m) is the required memory footprint. In the case of CPU-GPU hybrid computing [3], [4], in which only the m-step GMRES process is performed on GPU, this point was not discussed because GPU provides its own memory space; the total memory space increases. However, in the case of using only CPU, the matrix data $A^{(\mathrm{FP32})}$ in MP-GMRES(m) squeezes the memory consumption.

In MP-GMRES(m), the additional memory space is needed for $A^{(\mathrm{FP32})}$. On the other hand, changing from V_{m+1} to $V_{m+1}^{(\mathrm{FP32})}$ can reduce the memory consumption; V_{m+1} is not needed in MP-GMRES(m). In **Table 1**, the comparison on the memory footprint between FP64-GMRES(m) and MP-GMRES(m) is presented, where N_{nz} represents the number of nonzero elements in the matrix A. Here, the CRS (Compressed Row Storage) format, which generally consists of the three one-dimensional arrays `val`, `col_ind`, and `row_ptr`, is assumed for storing the matrix data; in this case, only the `val` in FP32 is additionally needed in MP-GMRES(m).

From Table 1, the change of the memory footprint from FP64-GMRES(m) is

$$4N_{\mathrm{nz}} - 4mn - 4n = 4n\left(\frac{N_{\mathrm{nz}}}{n} - (m+1)\right). \tag{10}$$

This result means that the memory footprint does not increase if the average of the number of nonzero elements per row (i.e., N_{nz}/n) in A is smaller than the restart frequency m, which is often satisfied in practical situations.

4.4 Discussion on the Expected Speedup

Finally, the expected speedup of MP-GMRES(m) over FP64-GMRES(m) is discussed. First, the execution time of the m-step Arnoldi process in FP32 and FP64 is roughly compared; for simplicity, the effect of cache memory is not considered at all in the following analysis. This process mainly consists of m SpMVs and MGS for m vectors, and it can be assumed that both are memory bounded computations.

In **Table 2**, the rough estimation of the memory access cost in the m-step Arnoldi process in FP64 and FP32 is summarized; for SpMV, memory access cost for two vectors are considered to-

Table 2 Rough estimation of the memory access cost (in bytes) in the m-step Arnoldi process in FP64 and FP32.

	FP64	FP32
SpMVs	$(12N_{\rm nz} + 20n)m$	$(8N_{\rm nz} + 12n)m$
MGS	$8 \cdot \frac{1}{2}m^2n$	$4 \cdot \frac{1}{2}m^2n$
Total	$4m^2n + 12mN_{\rm nz}$ $+20mn$	$2m^2n + 8mN_{\rm nz}$ $+12mn$

gether with that for the matrix stored in the CRS format. Let γ be the ratio of the number of repeating the m-step GMRES process in MP-GMRES(m) over that in FP64-GMRES(m); $\gamma \geq 1$ is generally expected. Then, ignoring the computational cost other than the repeated m-step Arnoldi processes, the speedup of MP-GMRES(m) over FP64-GMRES(m) is estimated as

$$\begin{aligned}(\text{Speedup}) &= \frac{4m^2n + 12mN_{\rm nz} + 20mn}{\gamma(2m^2n + 8mN_{\rm nz} + 12mn)} \\ &= \frac{2m + 6\frac{N_{\rm nz}}{n} + 10}{\gamma\left(m + 4\frac{N_{\rm nz}}{n} + 6\right)} \\ &\simeq \frac{2\rho + 6}{\gamma(\rho + 4)}, \end{aligned} \tag{11}$$

where $\rho = m/(N_{\rm nz}/n)$. From

$$1.5 < \frac{2\rho + 6}{\rho + 4} < 2, \tag{12}$$

the following observations are obtained:

- If $\gamma < 1.5$, the acceleration by MP-GMRES(m) is expected.
- If $\gamma = 1.0$, which means no degradation in the convergence behavior, the maximum speedup is up to 2.0, which is reasonable considering the ratio of bytes of FP64 and FP32.
- As ρ becomes larger, the expected speedup increases.

5. Numerical Experiments

Numerical experiments are conducted to clarify the mathematical behavior of MP-GMRES(m). For test matrices obtained from the SuiteSparse Matrix Collection [13], the number of total inner iterations for achieving a designated accuracy and the achievable accuracy within a designated limit of total inner iterations are investigated. The speedup ratio over FP64-GMRES(m) is also evaluated.

5.1 Experimental Settings

FP64-GMRES(m) and MP-GMRES(m) are implemented; all program codes are written in C. In the experiments on the number of total inner iterations and the achievable accuracy, sequential implementations are employed. For the evaluation of the speedup ratio, parallel implementations with OpenMP are used, in which the SpMV and MGS kernels in the Arnoldi process are parallelized. The CRS format is employed for storing the matrix data.

All experiments are carried out on a single computational node of the supercomputer system Grand Chariot operated in Hokkaido University. The information of this platform is listed in **Table 3**. The program codes are compiled by the Intel C compailer "`icc`" (ver. 19.1.3.304), and the options "`-xCORE-AVX512 -qopenmp`" are used for parallel implementations.

From the SuiteSparse Matrix Collection, matrices whose 2-norm condition number $\kappa_2(A)(= \|A\|_2/\|A^{-1}\|_2)$ is computed and

Table 3 Information of the computational platform used in the numerical experiments.

Item		Description
CPU	No.	Xeon Gold 6148
	# of cores	20
	Frequency	2.4 GHz
Node	# of CPUs	2
	Memory size	384 GB

in the range from $O(10^1)$ to $O(10^8)$ are selected; the information of the selected test matrices are listed in **Table 4**. The other settings in the experiments are as follows:

- Right-hand side vector: $b = [1, 1, \ldots, 1]^\top$
- Initial guess: $x_0 = [0, 0, \ldots, 0]^\top$
- Convergence criterion: $\dfrac{\|b - Ax\|_2}{\|b\|_2} \leq 10^{-10}$
- Iteration limit: (total # of inner iterations) $\leq n$
- Candidates of m: 50, 100, 200, 300, 400, 500

It is worth noting that all residuals shown in the subsequent tables and figures are computed with A and b in FP64 by FP64 arithmetic; there is no effect by FP32 in the evaluation of MP-GMRES(m), which is fair to that of FP64-GMRES(m).

5.2 Comparison of the Total Number of Inner Iterations

Table 5 shows the total number of inner iterations of FP64-GMRES(m) and MP-GMRES(m) for each matrix and each m. From this table, the following observations are obtained.

- Excepting a few cases (e.g., `TSOPF_RS_b39_c7`), if FP64-GMRES(m) attained the convergence criterion, MP-GMRES(m) also did. This indicates that if a problem can be solved by FP64-GMRES(m), it is expected that it will be solved also by MP-GMRES(m) although there is a possibility that more inner iterations will be required.
- There are some cases that FP64-GMRES(m) attained within one restart (i.e., less than m inner iterations), e.g., `cags10`, `Zhao1`. However, there are no such cases in MP-GMRES(m), and this is reasonable because the inner iterations in MP-GMRES(m) are performed in FP32.
- Excepting the above cases, if m is small (e.g., $m = 50, 100$), the total number of inner iterations of MP-GMRES(m) is almost equal to that of FP64-GMRES(m).
- It can be found that the difference of the total number of inner iterations between FP64-GMRES(m) and MP-GMRES(m) tends to increase as m becomes larger; as m grows, the total number of inner iterations of FP64-GMRES(m) basically decreases, but that of MP-GMRES(m) tends to increase.

For `memplus`, **Fig. 1** illustrates the change of the total number of inner iterations with m. As Fig. 1 shows, the total number of inner iterations first decreases and then increases in MP-GMRES(m), while it monotonically decreases in FP64-GMRES(m). This behavior is consistent with that mentioned in Section 4.2. It is also an interesting result that MP-GMRES(m) attained the convergence criterion with smaller total inner iterations than FP64-GMRES(m); however, clarifying its reason will be not easy.

Table 4 Information of the test matrices selected from the SuiteSparse Matrix Collection.

$\kappa_2(A)$	Name	n	N_{nz}	N_{nz}/n	Kind
$O(10^1)$	cage10	11,397	150,645	13	Directed weighted graph
$O(10^2)$	appu	14,000	1,853,104	132	Directed weighted random graph
	Zhao1	33,861	166,453	5	Electromagnetics problem
	FEM_3D_thermal1	17,880	430,740	24	Thermal problem
	ns3Da	20,414	1,679,599	82	Computational fluid dynamics problem
$O(10^3)$	poisson3Da	13,514	352,762	26	Computational fluid dynamics problem
	wang3	26,064	177,168	7	Semiconductor device problem
	epb1	14,734	95,053	6	Thermal problem
$O(10^4)$	coupled	11,341	98,523	9	Circuit simulation problem
	af23560	23,560	484,256	21	Computational fluid dynamics problem
	Zhao2	33,861	166,453	5	Electromagnetics problem
$O(10^5)$	memplus	17,758	126,150	7	Circuit simulation problem
	wang4	26,068	177,196	7	Semiconductor device problem
	viscoplastic2	32,769	381,326	12	Materials problem
$O(10^6)$	airfoil_2d	14,214	259,688	18	Computational fluid dynamics problem
	inlet	11,730	328,323	28	Model reduction problem
	jan99jac040sc	13,694	82,842	6	Economic problem
	chipcool1	20,082	281,150	14	Model reduction problem
$O(10^7)$	TSOPF_RS_b39_c7	14,098	252,446	18	Power network problem
	sme3Da	12,504	874,887	70	Structural problem
	garon2	13,535	390,607	29	Computational fluid dynamics problem
	shermanACb	18,510	145,149	8	2D/3D problem
$O(10^8)$	powersim	15,838	67,562	4	Power network problem
	circuit_3	12,127	48,137	4	Circuit simulation problem
	e40r0100	17,281	553,562	32	2D/3D problem
	rajat15	37,261	443,573	12	Circuit simulation problem

Table 5 Total number of inner iterations in FP64-GMRES(m) and MP-GMRES(m) for attaining the convergence criterion ($\|b - Ax\|_2/\|b\|_2 \leq 10^{-10}$): "—" represents that the method did not attain the convergence criterion by the iteration limit.

$\kappa_2(A)$	Matrix Name	$m = 50$		$m = 100$		$m = 200$		$m = 300$		$m = 400$		$m = 500$	
		FP64	MP	FP64	MP	FP64	MP	FP64	MP	FP64	MP	FP64	MP
$O(10^1)$	cage10	26	59	26	110	26	211	26	311	26	411	26	511
$O(10^2)$	appu	114	114	110	151	108	251	108	351	108	451	108	551
	Zhao1	43	73	43	123	43	223	43	323	43	423	43	523
	FEM_3D_thermal1	318	318	270	270	260	300	250	399	250	511	250	611
	ns3Da	2,310	2,316	1,983	1,984	1,993	1,989	1,791	1,790	1,945	1,945	1,522	1,522
$O(10^3)$	poisson3Da	306	320	260	270	184	446	184	651	184	833	184	1,018
	wang3	937	896	670	663	486	587	319	719	313	957	313	1,150
	epb1	1,994	1,881	1,781	1,786	1,516	1,517	1,290	1,302	1,153	1,153	1,095	1,272
$O(10^4)$	coupled	—	—	—	9,873	1,846	2,094	1,450	1,441	1,025	1,091	872	1,324
	af23560	—	—	—	—	—	—	4,799	4,777	4,305	4,283	4,442	4,351
	Zhao2	—	—	3,878	3,611	3,488	3,494	2,868	2,846	2,880	2,880	2,929	2,932
$O(10^5)$	memplus	5,900	4,521	3,030	2,943	1,942	2,044	1,532	2,404	1,459	3,265	1,352	3,108
	wang4	—	—	—	—	1,741	1,896	1,101	1,423	738	1,477	688	1,792
	viscoplastic2	—	—	2,061	1,969	1,530	1,658	1,409	1,700	1,375	2,046	1,299	2,274
$O(10^6)$	airfoil_2d	—	—	—	—	—	—	—	—	—	—	8,172	—
	inlet	—	—	—	—	—	—	—	—	—	—	—	—
	jan99jac040sc	—	—	—	—	—	—	—	—	—	—	—	—
	chipcool1	—	—	—	—	—	—	—	—	13,626	—	9,636	—
$O(10^7)$	TSOPF_RS_b39_c7	—	—	—	—	392	—	488	—	588	—	688	—
	sme3Da	—	—	—	—	—	—	—	—	—	—	—	—
	garon2	—	—	—	—	—	—	—	—	—	—	—	—
	shermanACb	—	—	—	—	—	—	—	—	—	—	—	—
$O(10^8)$	powersim	—	—	—	—	—	—	—	—	—	—	—	—
	circuit_3	—	—	—	—	—	—	—	—	—	—	—	—
	e40r0100	—	—	—	—	—	—	—	—	—	—	—	—
	rajat15	—	—	—	—	—	—	—	—	—	—	—	—

5.3 Comparison of the Achievable Accuracy

Table 6 provides the relative residual norm in FP64-GMRES(m) and MP-GMRES(m) at the iteration limit (or when the relative residual norm attained the convergence criterion). From this table, it is observed that when both the methods did not attain the convergence criterion, there is almost no difference in the final accuracy excepting the matrix `airfoil_2d`; almost no improvement from the initial guess was obtained in many cases ("-1" in the table means this). This fact indicates that regardless of using FP32, another technique, e.g., preconditioning, should be considered for solving these linear systems.

5.4 Case Studies with the Relative Residual Norm History

Among the results presented in the previous subsections, some interesting cases are analyzed in detail with the history of the relative residual norm; in the graphs of the history, a mark means a restart point, in which the residual norm is calculated explicitly by FP64 arithmetic.

Table 6 Achievable accuracy of FP64-GMRES(m) and MP-GMRES(m) within the limit of total inner iterations: $\lfloor \log_{10}(\|b - Ax\|_2/\|b\|_2) \rfloor$ when the solver attained the iteration limit or the convergence criterion ("-11" corresponds to this case) is listed.

$\kappa_2(A)$	Matrix Name	m = 50		m = 100		m = 200		m = 300		m = 400		m = 500	
		FP64	MP	FP64	MP	FP64	MP	FP64	MP	FP64	MP	FP64	MP
$O(10^1)$	cage10	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
$O(10^2)$	appu	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
	Zhao1	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
	FEM_3D_thermal1	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
	ns3Da	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
$O(10^3)$	poisson3Da	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
	wang3	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
	epb1	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
$O(10^4)$	coupled	-2	-2	-9	-11	-11	-11	-11	-11	-11	-11	-11	-11
	af23560	-1	-1	-1	-1	-1	-1	-11	-11	-11	-11	-11	-11
	Zhao2	-1	-1	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
$O(10^5)$	memplus	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
	wang4	-1	-1	-1	-1	-11	-11	-11	-11	-11	-11	-11	-11
	viscoplastic2	-1	-1	-11	-11	-11	-11	-11	-11	-11	-11	-11	-11
$O(10^6)$	airfoil_2d	-8	-3	-8	-3	-9	-4	-9	-3	-9	-5	-11	-9
	inlet	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	jan99jac040sc	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	chipcool1	-1	-1	-1	-2	-4	-9	-10	-3	-11	-6	-11	-4
$O(10^7)$	TSOPF_RS_b39_c7	-1	-1	-1	-1	-11	-1	-11	-1	-11	-1	-11	-1
	sme3Da	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	-1
	garon2	-1	-1	-1	-1	-2	-2	-3	-3	-3	-3	-3	-3
	shermanACb	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$O(10^8)$	powersim	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	circuit_3	-1	-1	-1	-1	-1	-1	-2	-2	-2	-2	-2	-2
	e40r0100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	rajat15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0

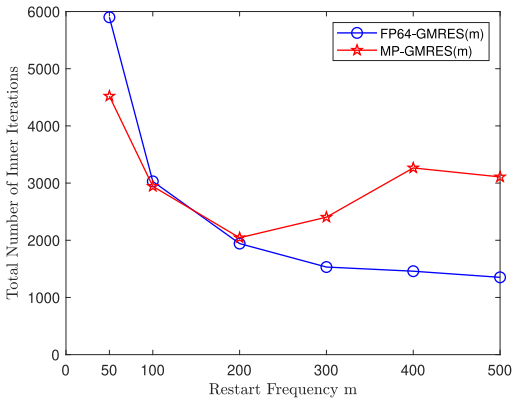
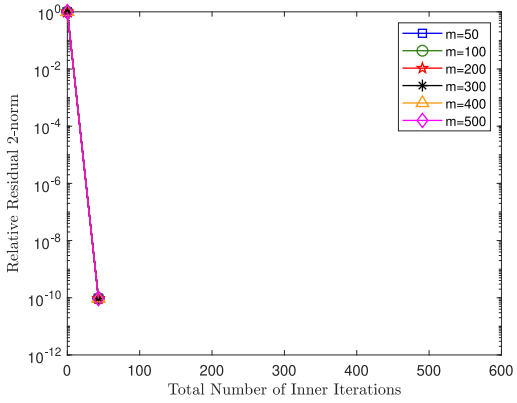
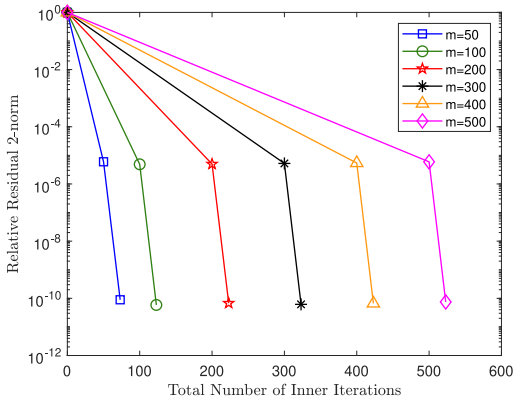


Fig. 1 The change of the total number of inner iterations in FP64-GMRES(m) and MP-GMRES(m) for `memplus`.

- `Zhao1` (**Fig. 2**): from Fig. 2 (b), it is found that no matter how large m is, the reduction of the residual 2-norm at the first restart point is limited (up to around 10^{-5}) in MP-GMRES(m), while FP64-GMRES(m) attained the convergence criterion within one restart as shown in Fig. 2 (a).
- `ns3Da` (**Fig. 3**): both FP64-GMRES(m) and MP-GMRES(m) provide behavior of the residual 2-norm history. The total number of inner iterations tends to be small as m grows in both methods.
- `memplus` (**Fig. 4**): as m increases, the total number of inner iterations decreases in FP64-GMRES(m) as shown in Fig. 4 (a). However, the total number of inner iterations first decreases and then increases (minimum is provided when m = 200) in MP-GMRES(m), which Fig. 4 (b) clearly illustrates. Another interesting fact is that when m = 50, the total number of inner iterations in MP-GMRES(m) is smaller than that in FP64-GMRES(m), whose reason is currently not known.
- `airfoil_2d` (**Fig. 5**): FP64-GMRES(m) attained the convergence criterion only when m = 500. Although MP-

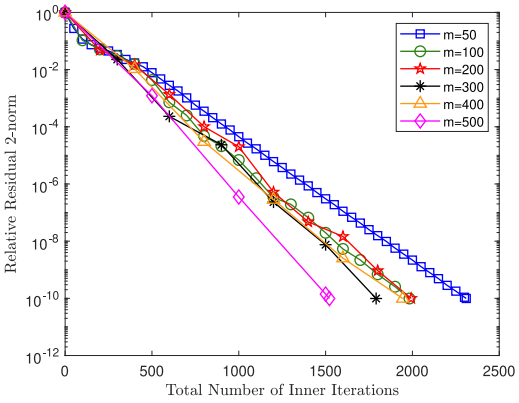


(a) FP64-GMRES(m)

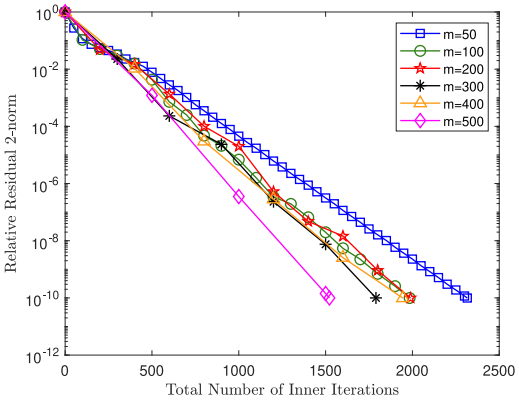


(b) MP-GMRES(m)

Fig. 2 Relative residual 2-norm history for `Zhao1`.

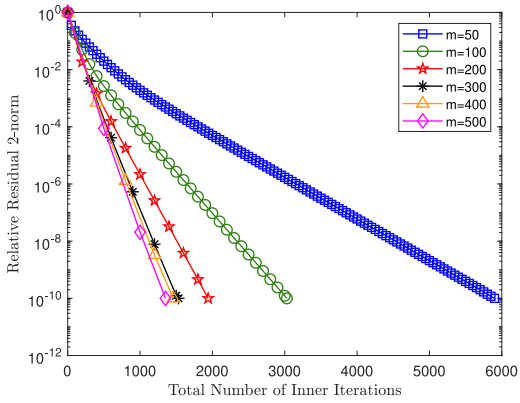


(a) FP64-GMRES(m)

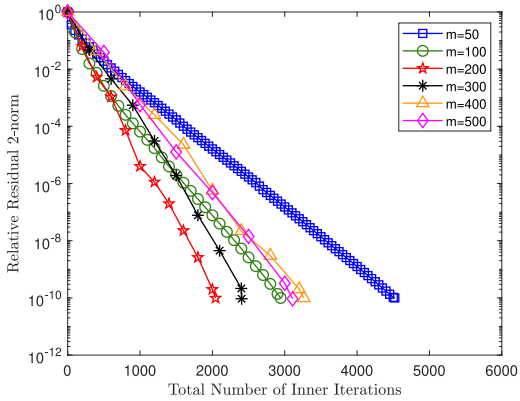


(b) MP-GMRES(m)

Fig. 3 Relative residual 2-norm history for `ns3Da`.



(a) FP64-GMRES(m)



(b) MP-GMRES(m)

Fig. 4 Relative residual 2-norm history for `memplus`.

GMRES(m) with $m = 500$ did not attain within the iteration limit, seen from Fig. 5 (b), it is expected to attain with more iterations. In both methods, the residual norm stagnates when $m \leq 400$, however, the achievable accuracy is different.

- `TSOPF_RS_b39_c7` (**Fig. 6**): FP64-GMRES(m) attained the convergence criterion when $m \geq 200$, while almost no decrease of the residual norm is found in MP-GMRES(m) with any m. The behavior of the residual norm is totally different between FP64-GMRES(m) and MP-GMRES(m).

5.5 Evaluation of the Execution Time

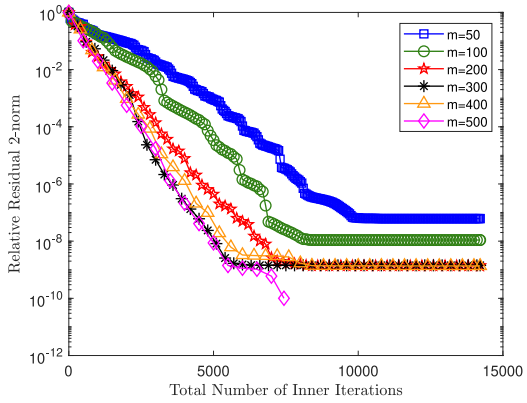
Table 7 presents the execution time of FP64-GMRES(m) and MP-GMRES(m) in multi-threaded execution using 40 threads; we used the default setting on the thread affinity: `scatter`. Since the main interest in this paper is the numerical behavior of the methods, the results in this table should be regarded as reference data; there is a room to further tune our implementation (e.g., using other storage format than CRS and optimizing thread parallelization) for improving the performance. It is also worth mentioning that the characteristics of the execution time of SpMV depend on the size of a matrix due to the effect of the cache memory; only small matrices are tested in our experiments.

From Table 7, it is confirmed that the execution time tends to be short when m is small in both FP64-GMRES(m) and MP-GMRES(m), which is a well-known fact among the users of the GMRES(m) method. Since the total number of inner iterations is almost equivalent between FP64-GMRES(m) and MP-GMRES(m) when m is small, which is shown in Table 5, MP-GMRES(m) outperforms FP64-GMRES(m) in terms of the execution time for many test matrices. Even in the case that larger m reduces the total number of inner iterations in FP64-GMRES(m), e.g., `wang4`, MP-GMRES(m) is still faster, in which about 2.5 times the total inner iterations are required: comparison between MP-GMRES(200) and FP64-GMRES(400).

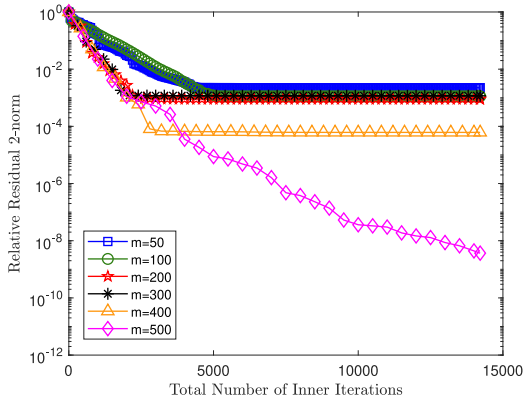
Among the results in **Table 8**, for the cases in which the setting of m that provides the shortest execution time is not different between FP64-GMRES(m) and MP-GMRES(m), the actual speedup ratio of MP-GMRES(m) over FP64-GMRES(m) is compared with the estimated speedup ratio obtained by Eq. (11); γ in Eq. (11) is calculated by the actual number of the iterations. The results of the comparison are given in Table 8. Excepting a few matrices, i.e., `appu`, `ns3Da`, `poisson3Da`, the estimated ratios seem to be reasonable because the estimation is based on the rough modeling. In the above exceptions, ρ is small compared with other cases, which means that the cost of SpMVs is relatively large to that of MGS. Since the modeling of the execution time of SpMV is more complicated than that for MGS, this may be one of reasons for the large gap between the actual and estimated speedup ratio.

5.6 Summary of the Numerical Experiments

From the presented numerical results, it is confirmed that if

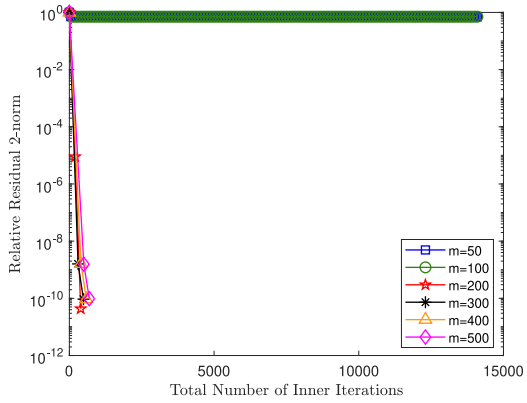


(a) FP64-GMRES(m)

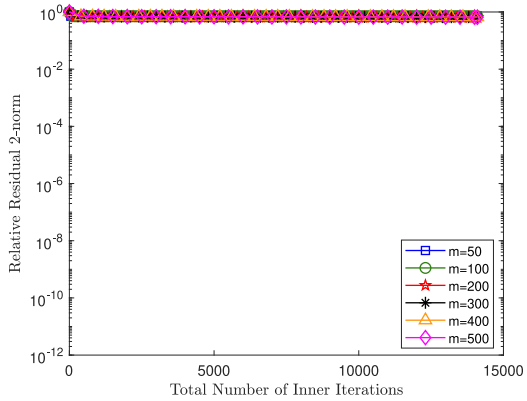


(b) MP-GMRES(m)

Fig. 5 Relative residual 2-norm history for `airfoil_2d`.



(a) FP64-GMRES(m)



(b) MP-GMRES(m)

Fig. 6 Relative residual 2-norm history for `TSOPF_RS_b39_c7`.

a problem is solved by FP64-GMRES(m), MP-GMRES(m) also can solve it excepting a few matrices. This fact is important for the purpose of using MP-GMRES(m) in practical applications as an alternative of FP64-GMRES(m). For problems that have been solved by FP64-GMRES(m), it is worth considering the use of MP-GMRES(m).

In the case of employing small m, the total number of inner iterations for attaining the convergence criterion is almost equivalent between FP64-GMRES(m) and MP-GMRES(m). However, when m is larger, the total number of inner iterations basically decreases in FP64-GMRES(m) but increases in MP-GMRES(m), and there is a significant difference between them. Although we have some perspectives for the phenomena based on the history of the residual 2-norm, further investigations are needed for clarifying its detailed mechanism. Excepting the case that a problem is solved only when using large m, basically small m provides the fastest execution time. As mentioned above, in such a case, both methods need almost equivalent total inner iterations, and the time per inner iteration is smaller in MP-GMRES(m) than in FP64-GMRES(m) on a standard CPU platform. Thus, it is expected that MP-GMRES(m) outperforms FP64-GMRES(m) in terms of the execution time.

As mentioned in Section 2, there are several papers that study the MP-GMRES(m) method [3], [4], [25], [26], [34]. Among them, for the papers [3], [4], [25], [34] [*3], we briefly compare our results with those presented in these papers. First, our results are consistent with the previous studies in the following points:

- If a problem is solved by FP64-GMRES(m), basically it can be solved also by MP-GMRES(m).
- When m is small, the total number of inner iterations of MP-GMRES(m) is almost equivalent to that in FP64-GMRES(m); cf., the results with $m = 10$ reported in the papers [3], [34].
- For some problems, MP-GMRES(m) can outperform FP64-GMRES(m) in terms of the execution time; cf., the results presented in the papers [3], [4], [25], [34].
- If a problem is solved within single restart by FP64-GMRES(m), MP-GMRES(m) requires more inner iterations (e.g., `cage10` in our experiments); cf., the results provided in the paper [25].

Next, we clarify insights that have been newly obtained through our investigation:

- The results on the achieved accuracy are reported including those in the case that the criterion is not satisfied. These results indicate that the attainable accuracy will be basically equivalent between MP-GMRES(m) and FP64-GMRES(m).
- The comprehensive results for different settings of m are presented. From them, the differences in the tendency of the total number of inner iterations between MP-GMRES(m) and FP64-GMRES(m) are observed; the tendency differs between small m and large m.

[*3] Since the paper [26] deals with only the preconditioned method, we do not refer to it.

Table 7 Execution time of FP64-GMRES(m) and MP-GMRES(m) in multi-threaded execution using 40 threads: the result with underline is the fastest among different settings of m, and "—" means that the method did not attain the convergence criterion.

$\kappa_2(A)$	Matrix Name	Method	$m = 50$	$m = 100$	$m = 200$	$m = 300$	$m = 400$	$m = 500$
$O(10^1)$	cage10	FP64	1.03×10^{-2}	1.22×10^{-2}	1.10×10^{-2}	1.25×10^{-2}	$\underline{9.49 \times 10^{-3}}$	1.14×10^{-2}
		MP	$\underline{1.43 \times 10^{-2}}$	3.40×10^{-2}	9.85×10^{-2}	2.01×10^{-1}	4.09×10^{-1}	5.43×10^{-1}
$O(10^2)$	appu	FP64	$\underline{4.34 \times 10^{-2}}$	6.31×10^{-2}	6.70×10^{-2}	7.36×10^{-2}	6.80×10^{-2}	7.58×10^{-2}
		MP	$\underline{4.58 \times 10^{-2}}$	8.33×10^{-2}	1.72×10^{-1}	2.96×10^{-1}	4.78×10^{-1}	7.27×10^{-1}
	Zhao1	FP64	3.38×10^{-2}	3.77×10^{-2}	3.72×10^{-2}	3.79×10^{-2}	$\underline{3.27 \times 10^{-2}}$	3.49×10^{-2}
		MP	$\underline{2.90 \times 10^{-2}}$	7.75×10^{-2}	3.03×10^{-1}	7.62×10^{-1}	1.39×10^{0}	2.18×10^{0}
	FEM_3D_thermal1	FP64	$\underline{1.08 \times 10^{-1}}$	1.69×10^{-1}	3.05×10^{-1}	4.64×10^{-1}	4.53×10^{-1}	4.78×10^{-1}
		MP	$\underline{6.36 \times 10^{-2}}$	9.29×10^{-2}	1.71×10^{-1}	3.24×10^{-1}	6.00×10^{-1}	9.12×10^{-1}
	ns3Da	FP64	$\underline{1.21 \times 10^{0}}$	1.74×10^{0}	3.45×10^{0}	5.33×10^{0}	7.70×10^{0}	7.89×10^{0}
		MP	$\underline{9.59 \times 10^{-1}}$	1.15×10^{0}	1.77×10^{0}	2.34×10^{0}	3.49×10^{0}	3.62×10^{0}
$O(10^3)$	poisson3Da	FP64	$\underline{8.94 \times 10^{-2}}$	1.29×10^{-1}	1.71×10^{-1}	1.65×10^{-1}	1.69×10^{-1}	1.80×10^{-1}
		MP	$\underline{8.32 \times 10^{-2}}$	9.35×10^{-2}	2.34×10^{-1}	4.44×10^{-1}	8.43×10^{-1}	1.34×10^{0}
	wang3	FP64	$\underline{4.39 \times 10^{-1}}$	6.45×10^{-1}	9.65×10^{-1}	1.15×10^{0}	1.26×10^{0}	1.35×10^{0}
		MP	$\underline{2.35 \times 10^{-1}}$	3.20×10^{-1}	4.81×10^{-1}	9.35×10^{-1}	1.81×10^{0}	2.79×10^{0}
	epb1	FP64	$\underline{6.09 \times 10^{-1}}$	9.65×10^{-1}	1.59×10^{0}	2.19×10^{0}	2.81×10^{0}	3.50×10^{0}
		MP	$\underline{3.24 \times 10^{-1}}$	5.19×10^{-1}	8.09×10^{-1}	9.96×10^{-1}	1.24×10^{0}	1.50×10^{0}
$O(10^4)$	coupled	FP64	—	—	$\underline{1.45 \times 10^{0}}$	1.77×10^{0}	1.66×10^{0}	1.90×10^{0}
		MP	—	—	$\underline{8.65 \times 10^{-1}}$	9.10×10^{-1}	8.87×10^{-1}	1.26×10^{0}
	af23560	FP64	—	—	—	$\underline{1.47 \times 10^{1}}$	1.78×10^{1}	2.46×10^{1}
		MP	—	—	—	$\underline{5.89 \times 10^{0}}$	7.35×10^{0}	9.91×10^{0}
	Zhao2	FP64	—	$\underline{5.12 \times 10^{0}}$	1.08×10^{1}	1.45×10^{1}	1.93×10^{1}	2.48×10^{1}
		MP	—	$\underline{2.44 \times 10^{0}}$	4.36×10^{0}	6.51×10^{0}	9.06×10^{0}	1.20×10^{1}
$O(10^5)$	memplus	FP64	2.32×10^{0}	$\underline{2.12 \times 10^{0}}$	2.69×10^{0}	3.49×10^{0}	4.36×10^{0}	5.16×10^{0}
		MP	1.30×10^{0}	$\underline{1.23 \times 10^{0}}$	1.30×10^{0}	2.18×10^{0}	3.37×10^{0}	4.14×10^{0}
	wang4	FP64	—	—	3.53×10^{0}	3.62×10^{0}	$\underline{3.34 \times 10^{0}}$	3.65×10^{0}
		MP	—	—	$\underline{1.69 \times 10^{0}}$	2.00×10^{0}	2.85×10^{0}	4.55×10^{0}
	viscoplastic2	FP64	—	$\underline{3.12 \times 10^{0}}$	4.69×10^{0}	6.67×10^{0}	8.89×10^{0}	1.01×10^{1}
		MP	—	$\underline{1.59 \times 10^{0}}$	2.22×10^{0}	3.17×10^{0}	4.68×10^{0}	6.60×10^{0}

Table 8 Comparison of the speedup ratio of MP-GMRES(m) over FP64-GMRES(m) between the actual and estimated results: for the cases in Table 7 in which the setting of m that provides the shortest execution time is not different between FP64-GMRES(m) and MP-GMRES(m), the actual speedup ratio is compared with the estimation obtained by Eq. (11), in which γ is calculated by the actual number of the iterations.

$\kappa_2(A)$	Matrix Name	m	N_{nz}/n	ρ	γ	Actual Speedup	Estimated Speedup
$O(10^2)$	appu	50	132	0.38	1.00	0.95	1.54
	FEM_3D_thermal1	50	24	2.08	1.00	1.70	1.67
	ns3Da	50	82	0.61	1.00	1.26	1.56
$O(10^3)$	poisson3Da	50	26	1.92	1.07	1.08	1.56
	wang3	50	7	7.14	0.96	1.87	1.90
	epb1	50	6	8.33	0.94	1.88	1.94
$O(10^4)$	coupled	200	9	22.22	1.06	1.68	1.82
	af23560	300	21	14.29	1.00	2.50	1.90
	Zhao2	100	5	20.00	0.97	2.10	1.97
$O(10^5)$	memplus	100	7	14.29	1.02	1.73	1.86
	viscoplastic2	100	12	8.33	0.97	1.96	1.90

- Typical histories of the relative residual 2-norm are shown. They clearly illustrate the behaviour of MP-GMRES(m) and FP64-GMRES(m) and help us to better understand the characteristics of MP-GMRES(m) and further improve the method.
- The comparison between the obtained and estimated speedup ration is provided. Although the estimation is based on the rough model, this comparison supports the obtained timing results.

6. Conclusion

MP-GMRES(m), the mixed precision variant of the GMRES(m) method using FP64 and FP32, was investigated through the numerical experiments and compared with FP64-GMRES(m), the traditional GMRES(m) method using only FP64. For various test matrices, with different settings of m, the numerical behaviors, namely the total number of inner iterations and the achievable accuracy, were examined. Among the obtained results, typical cases were analyzed in detail with the residual 2-norm history. The execution time of the methods

was also evaluated on a standard CPU platform.

The main observations obtained though the numerical experiments in this paper are as follows.

- If a problem is solved by FP64-GMRES(m), also MP-GMRES(m) can basically solve it; only a few exceptions were found.
- When m is small, the total number of inner iterations for attaining the convergence criterion is almost equivalent between FP64-GMRES(m) and MP-GMRES(m).
- As m grows, the total number of inner iterations often decreases in FP64-GMRES(m) but tends to increase in MP-GMRES(m); detailed reasons for it are currently not clear.
- Some interesting phenomena about history of the residual 2-norm were observed, and further investigations for them will be helpful for better understanding of MP-GMRES(m).
- On the standard CPU platform, it can be expected that MP-GMRES(m) outperforms FP64-GMRES(m) in terms of the execution time.

The presented results in this paper are consistent with those already reported in the existing papers [3], [4], [25], [34] and provide new insights on the numerical behavior of MP-GMRES(m), which will be helpful both for the use in applications and the further improvement of the method.

A main issue remained to be investigated is clarifying the mechanism and reasons for the different behavior of the total number of inner iterations when m becomes large. One possibility is mentioned in Section 4.2: the perturbation for the minimization problem solved in the m-step GMRES process. Through numerical experiments, its impact on the total number of inner iterations should be investigated. In addition, based on the results of such investigations, designing a way of determining an appropriate setting of m in MP-GMRES(m) will be crucial for the efficient use in applications, as that presented in the paper [25].

Some techniques for improving the convergence of the GMRES(m) method have been proposed: adaptively determining m [6], [35] and modifying the approximate solution at restart [21], [22]. It will be interesting to apply these techniques to MP-GMRES(m) and examine their efficiency; it is currently not clear whether similar performance improvement will be obtained as in FP64-GMRES(m).

Finally, investigating the case of using a preconditioner is another important task. As shown in the numerical results in this paper, for a difficult problem, both FP64-GMRES(m) and MP-GMRES(m) cannot obtain a solution with required accuracy, and a preconditioner is necessary. A mixed precision variant of the preconditioned GMRES(m) is already examined [25], [26], however, its numerical aspects such as presented in this paper have been still not clear. In addition, there is a large design space of mixed precision algorithms. Thus, detailed investigations will be important for developing a efficient mixed precision variant of the preconditioned GMRES(m) method.

Acknowledgments The authors thank the anonymous referees for their valuable comments. This research was supported by JSPS KAKENHI Grant Numbers JP19H04122 and JP19H05662, and JST, PRESTO Grant Number JPMJPR20M8, Japan. This research was also supported by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" and "High Performance Computing Infrastructure" in Japan (Project ID: jh210015-NAH). The first author was supported by Hokkaido University Ambitious Doctoral Fellowship (Information Science and AI).

References

[1] Abdelfattah, A., Anzt, H., Boman, E.G., Carson, E., Cojean, T., Dongarra, J., Fox, A., Gates, M., Higham, N.J., Li, X.S., Loe, J., Luszczek, P., Pranesh, S., Rajamanickam, S., Ribizel, T., Smith, B.F., Swirydowicz, K., Thomas, S., Tomov, S., Tsai, Y.M. and Yang, U.M.: A survey of numerical linear algebra methods utilizing mixed-precision arithmetic, *The International Journal of High Performance Computing Applications*, Vol.35, No.4, pp.344–369 (2021).

[2] Angerer, C.M., Polig, R., Zegarac, D., Giefers, H., Hagleitner, C., Bekas, C. and Curioni, A.: A fast, hybrid, power-efficient high-precision solver for large linear systems based on low-precision hardware, *Sustainable Computing: Informatics and Systems*, Vol.12, pp.72–82 (2016).

[3] Anzt, H., Heuveline, V. and Rocker, B.: An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations, *High Performance Computing for Computational Science – VECPAR 2010*, pp.58–70 (2011).

[4] Anzt, H., Heuveline, V. and Rocker, B.: Mixed Precision Iterative Refinement Methods for Linear Systems: Convergence Analysis Based on Krylov Subspace Methods, *Applied Parallel and Scientific Computing*, pp.237–247 (2012).

[5] Baboulin, M., Buttari, A., Dongarra, J., Kurzak, J., Langou, J., Langou, J., Luszczek, P. and Tomov, S.: Accelerating scientific computations with mixed precision algorithms, *Computer Physics Communications*, Vol.180, No.12, pp.2526–2533 (2009).

[6] Baker, A.H., Jessup, E.R. and Kolev, T.V.: A simple strategy for varying the restart parameter in GMRES(m), *Journal of Computational and Applied Mathematics*, Vol.230, No.2, pp.751–761 (2009).

[7] Blanchard, P., Higham, N.J., Lopez, F., Mary, T. and Pranesh, S.: Mixed Precision Block Fused Multiply-Add: Error Analysis and Application to GPU Tensor Cores, *SIAM Journal on Scientific Computing*, Vol.42, No.3, pp.C124–C141 (2020).

[8] Buttari, A., Dongarra, J., Kurzak, J., Luszczek, P. and Tomov, S.: Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy, *ACM Trans. Mathematical Software*, Vol.34, No.4, pp.17:1–17:22 (2008).

[9] Buttari, A., Dongarra, J., Langou, J., Langou, J., Luszczek, P. and Kurzak, J.: Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems, *The International Journal of High Performance Computing Applications*, Vol.21, No.4, pp.457–466 (2007).

[10] Carson, E. and Higham, N.J.: A New Analysis of Iterative Refinement and Its Application to Accurate Solution of Ill-Conditioned Sparse Linear Systems, *SIAM Journal on Scientific Computing*, Vol.39, No.6, pp.A2834–A2856 (2017).

[11] Carson, E. and Higham, N.J.: Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions, *SIAM Journal on Scientific Computing*, Vol.40, No.2, pp.A817–A847 (2018).

[12] Carson, E., Higham, N.J. and Pranesh, S.: Three-Precision GMRES-Based Iterative Refinement for Least Squares Problems, *SIAM Journal on Scientific Computing*, Vol.42, No.6, pp.A4063–A4083 (2020).

[13] Davis, T.A. and Hu, Y.: The University of Florida Sparse Matrix Collection, *ACM Trans. Mathematical Software*, Vol.38, No.1, pp.1:1–1:25 (2011).

[14] Demmel, J.W.: *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics (1997).

[15] Georgescu, S. and Okuda, H.: Automatically Tuned Mixed-Precision Conjugate Gradient Solver, *Software Automatic Tuning: From Concepts to State-of-the-Art Results*, pp.103–119, Springer (2010).

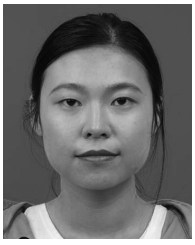
[16] Haidar, A., Tomov, S., Dongarra, J. and Higham, N.J.: Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers, *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp.603–613 (2018).

[17] Higham, N.J.: *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, second edition (2002).

[18] Higham, N.J. and Pranesh, S.: Exploiting Lower Precision Arithmetic in Solving Symmetric Positive Definite Linear Systems and Least Squares Problems, *SIAM Journal on Scientific Computing*, Vol.43, No.1, pp.A258–A277 (2021).

[19] Higham, N.J., Pranesh, S. and Zounon, M.: Squeezing a Matrix into Half Precision, with an Application to Solving Linear Systems, *SIAM*

Journal on Scientific Computing, Vol.41, No.4, pp.A2536–A2551 (2019).
[20] Ikuno, S., Kawaguchi, Y., Fujita, N., Itoh, T., Nakata, S. and Watanabe, K.: Iterative Solver for Linear System Obtained by Edge Element: Variable Preconditioned Method With Mixed Precision on GPU, *IEEE Trans. Magnetics*, Vol.48, No.2, pp.467–470 (2012).
[21] Imakura, A., Sogabe, T. and Zhang, S.-L.: An Efficient Variant of the GMRES(*m*) Method Based on the Error Equations, *East Asian Journal on Applied Mathematics*, Vol.2, No.1, pp.19–32 (2012).
[22] Imakura, A., Sogabe, T. and Zhang, S.-L.: A Look-Back-type restart for the restarted Krylov subspace methods for solving non-Hermitian linear systems, *Japan Journal of Industrial and Applied Mathematics*, Vol.35, No.2, pp.835–859 (2018).
[23] Kudo, S., Nitadori, K., Ina, T. and Imamura, T.: Implementation and Numerical Techniques for One EFlop/s HPL-AI Benchmark on Fugaku, *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems* (*ScalA*), pp.69–76 (2020).
[24] Langou, J., Langou, J., Luszczek, P., Kurzak, J., Buttari, A. and Dongarra, J.: Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy (Revisiting Iterative Refinement for Linear Systems), *SC '06: Proc. 2006 ACM/IEEE Conference on Supercomputing*, p.50 (2006).
[25] Lindquist, N., Luszczek, P. and Dongarra, J.: Improving the Performance of the GMRES Method Using Mixed-Precision Techniques, *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI*, pp.51–66 (2020).
[26] Lindquist, N., Luszczek, P. and Dongarra, J.: Accelerating Restarted GMRES with Mixed Precision Arithmetic, *IEEE Trans. Parallel and Distributed Systems*, Vol.33, pp.1027–1037 (2022).
[27] Loe, J.A., Glusa, C.A., Yamazaki, I., Boman, E.G. and Rajamanickam, S.: Experimental Evaluation of Multiprecision Strategies for GMRES on GPUs, *2021 IEEE International Parallel and Distributed Processing Symposium Workshops* (*IPDPSW*), pp.469–478 (2021).
[28] Ogita, T.: Accurate Matrix Factorization: Inverse LU and Inverse QR Factorizations, *SIAM Journal on Matrix Analysis and Applications*, Vol.31, No.5, pp.2477–2497 (2010).
[29] Ogita, T. and Aishima, K.: Iterative refinement for symmetric eigenvalue decomposition, *Japan Journal of Industrial and Applied Mathematics*, Vol.35, No.3, pp.1007–1035 (2018).
[30] Ogita, T. and Aishima, K.: Iterative refinement for symmetric eigenvalue decomposition II: Clustered eigenvalues, *Japan Journal of Industrial and Applied Mathematics*, Vol.36, No.2, pp.435–459 (2019).
[31] Saad, Y. and Schultz, M.H.: GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM Journal on Scientific and Statistical Computing*, Vol.7, No.3, pp.856–869 (1986).
[32] Saad, Y.: *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics (2003).
[33] Tadano, H. and Sakurai, T.: On Single Precision Preconditioners for Krylov Subspace Iterative Methods, *Large-Scale Scientific Computing*, pp.721–728 (2008).
[34] Turner, K. and Walker, H.F.: Efficient High Accuracy Solutions with GMRES(*m*), *SIAM Journal on Scientific and Statistical Computing*, Vol.13, No.3, pp.815–825 (1992).
[35] Zhang, L. and Nodera, T.: A new adaptive restart for GMRES(*m*) method, *Australian and New Zealand Industrial and Applied Mathematics Journa*, Vol.46, pp.C409–C425 (2005).



Yingqi Zhao was born in 1995. She received her B.S. degree in Information and Computation Science, and Master of Natural Science degree in Computational Mathematics from Ocean University of China in 2017 and 2020, respectively. She is currently working towards her Ph.D degree in Graduate School of Information Science and Technology, Hokkaido University. Her research interests include high performance computing and numerical linear iterative solver.

Takeshi Fukaya was born in 1983. He received his B.E, M.E., and Ph.D. from Nagoya University in 2007, 2009, and 2012, respectively. He worked in Kobe University and RIKEN AICS, and became an assistant professor in the Information Initiative Center, Hokkaido University in 2015. He also worked as JST PRESTO researcher in November, 2020-March, 2022. His major research interests include high performance computing and numerical linear algebra. He is a member of IPSJ, JSIAM, and SIAM.

Linjie Zhang was born in 1975. She received her B.E. degree from Xi'an Jiaotong University in 1997, M.E., and Ph.D. degrees from Keio University in 2004 and 2007, respectively. She works as an associate professor in Ocean University of China. Her research interests include high performance computing, linear iterative solver and image segamentation.

Takeshi Iwashita was born in 1971. He received a B.E., an M.E., and a Ph.D. from Kyoto University in 1992, 1995, and 1998, respectively. In 1998–1999, he worked as a post-doctoral fellow of the JSPS project in the Graduate School of Engineering, Kyoto University. He moved to the Data Processing Center of the same university in 2000. In 2003–2014, he worked as an associate professor in the Academic Center for Computing and Media Studies, Kyoto University. He currently works as a professor in the Information Initiative Center, Hokkaido University. His research interests include high performance computing, linear iterative solver, and electromagnetic field analysis. He is a member of IEEE, SIAM, IPSJ, IEEJ, JSIAM, JSCES, and JSAEM.