A Study on Mixed Precision Iterative Refinement using Low Precision Krylov Methods

Yingqi ZHAO

Graduate School of Information Science and Technology,

Hokkaido University

February, 2024

Preface

Numerical linear algebra has wide applications in many aspects of life and is at the heart of important computational aspects of practical application problems. Numerical methods, often used to solve linear algebra problems, e.g., linear systems, eigenvalue problems, singular value decompositions, etc., in engineering and information sciences, play a key role and attract much attention. Among them, linear solvers for solving linear systems $Ax = b$ are of vital importance in scientific computation and their computational time usually attracts much attention. An efficient linear solver can help accelerate practical applications. Therefore, the optimization of numerical methods and linear solvers is essential.

Double-precision floating-point (also called FP64) has been regarded as a standard in most scientific computation for a long time. However, due to the reason of power budgets, the performance of FP64 has become more and more challenging and difficult to improve. Within recent years, some newly appeared hardware shows potentially high ability of low precision computing, some of which are equipped with specified units that can provide great performance in low precision computing, and have already been used in practical applications that do not strictly require high precision computing and accept low precision computing, i.e., machine learning. These circumstances promote the development of low precision computing while the output accuracy needs to be guaranteed. To handle this issue, one important strategy is to develop mixed precision (MP) algorithms that efficiently combine different precisions and achieve the same accuracy as the traditional method using only FP64.

In this study, I focus on linear solvers and develop mixed precision algorithms for solving linear systems $Ax = b$ whose coefficient matrix A is large, sparse, and non-symmetric. For this problem, the Krylov subspace methods are widely used in the field of numerical linear algebra. Some studies on mixed precision computing for the Krylov subspace methods have already been reported. However, there are still many issues that need to be investigated. This study aims to provide new insights into mixed precision computing using the Krylov subspace methods, which will contribute to improving the computation performance of various applications that need liner solvers. In this study, two typical Krylov subspace methods, which are commonly used to solve non-symmetric linear systems, are considered. One is GMRES(m) method, and the other is BiCGSTAB method. Based on these two algorithms, I develop two mixed precision algorithms and conduct comprehensive numerical experiments using 26 test matrices selected

from the SuiteSparse Matrix Collection. Through numerical experiments, I investigate the numerical characteristics and evaluate the effectiveness of the developed mixed precision algorithms in detail.

In Chapter 3, a mixed precision variant of the GMRES(m) method using FP64 and FP32, which is called MP-GMRES(m), is investigated. Using the number of the inner iterations as restart frequency m, I have studied its numerical behavior through comprehensive experiments with different m from both theoretical and practical aspects. A detailed comparison with the traditional GMRES(m) method using only FP64 is also made. Detailed analysis and comparison of the obtained results are given from the following three aspects: the maximum attainable accuracy, the number of iterations, and the execution time. From the obtained results, MP-GMRES(m) has almost the same problem-solving ability as GMRES(m), and there is almost no difference in the final attainable accuracy if both two algorithms can not solve the problems. Although MP-GMRES(m) requires more number of iterations than GMRES(m), it provides a shorter execution time for most cases. I have also found some differences between these two methods; for example, as m increases, the number of iterations tends to decrease in GMRES(m) while increasing in MP-GMRES(m).

In Chapter 4, a mixed precision variant of iterative refinement using BiCGSTAB algorithm (MP-IR using BiCGSTAB) is developed and investigated in detail. In this algorithm, low precision(FP32) BiCGSTAB is employed as an inner solver in mixed precision iterative refinement, and two approaches are used for determining the restart: the number of the inner iterations m and the decrease of the residual 2-norm ϵ in the inner BiCGSTAB loop. Several sets of experiments are conducted, and the obtained results are analyzed and compared with the traditional BiCGSTAB method, as well as the MP-GMRES(m) method, which is studied in Chapter 3. The experiment results show that MP-IR using BiCGSTAB sometimes outperforms MP-GMRES(m) and is the fastest among all methods, especially for problems with small condition numbers, although MP-IR using BiCGSTAB is sensitive to a target problem. Similar to Chapter 3, comprehensive experimental results on the maximum attainable accuracy, the number of iterations, and the execution time are also provided and analyzed in this chapter.

The rest of the thesis is organized as follows. In Chapter 1, I introduce the research background, purposes, and main contributions. The structure of the thesis is also briefly described. Chapter 2 summarizes the related work, including mixed precision computing, the Krylov subspace methods, and some recent research on applying mixed precision computing to the Krylov subspace methods. Chapter 5 provides an overall summary of my research and discusses future work.

In summary, the thesis aims to investigate the numerical characteristics and effectiveness of the mixed precision algorithms and show the applicability of mixed precision computing in numerical linear solvers. This study will provide options to accelerate the applications that need linear solvers and give some new insights into using the high ability of low precision computing, which hopes to promote further development of mixed precision linear solvers.

Acknowledgment

First, I would like to express my sincere thanks to everyone who ever helped me with this thesis.

I would like to express my sincere gratitude to my supervisors, Prof. Iwashita and Prof. Fukaya, for helping me greatly in my research. They have supported me with patient guidance, inspiration, and encouragement, from the selection of the research topic, research idea, and experiment demonstration to thesis revision. When I encountered difficulties during my research, their kind guidance and suggestions gave me great encouragement. It is my great honor and joy to study under their supervision. Furthermore, I have also benefited from their personality and diligence, which I will cherish my whole life.

I also would like to express my sincere gratitude to my master's supervisor, Prof. Zhang Linjie. Under her patient guidance, I developed a strong interest in academic research. And thanks to her introduction, I had the opportunity to study my Ph.D. course in my current laboratory. Her attitudes towards research and life have deeply influenced me.

I also greatly appreciate Prof. Munetomo and Prof. IIda for their suggestions on revising the draft of this thesis, as well as for pointing out the problems that should be addressed during the thesis revision and presentation. With their suggestions and comments, I could further clarify my thoughts on the revision of the thesis and was able to improve my thesis.

In addition, I would also like to thank Prof. Iwashita, Prof. Fukaya, and other faculty staff for helping me with aspects of my life in Japan. I would also like to express my deep gratitude to the university for providing scholarship support. Their help made me feel at ease in a new environment and allowed me to devote myself to my research in a foreign country. I also thank my fellow laboratory students, who created a strong academic atmosphere that positively impacted the research.

Finally, I would also like to express my heartfelt thanks and love to my family and friends, who gave me a lot of financial support and encouragement and whom I love and care about.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Background

Linear algebra has a wide range of applications in many aspects of life and is the core of critical computational aspects of practical application problems. In engineering applications, linear algebra can help us better simulate and analyze when dealing with complex mathematical models and huge amounts of data. In information science applications, we can understand and even optimize algorithms through linear algebra. Even in the fields of finance, medicine, and other fields, linear algebra has significant applications, such as price prediction, medical modeling, and so on. In many practical applications, numerical methods have been frequently used to solve linear algebra problems, such as systems of linear equations, eigenvalue problems, singular value decomposition, etc. An excellent numerical method can help us improve the computational efficiency. Among them, linear solvers, which solve systems of linear equations, are one of the most important building blocks in scientific computing. It has an impact on both the quality and time of problem-solving and is the bottleneck of the computational process. Therefore, it is crucial to find more and more efficient linear solvers to accelerate application programs.

Traditionally, double-precision floating-point number (FP64 or float64) has been widely used as a standard in scientific computations, especially in applications requiring standard precision scientific computation, engineering simulations, etc. In general, Floating-point Operations Per Second, which is called FLOPS for short, has been used to evaluate the performance of computer systems. However, due to the limitation of the power budget, the performance of FP64 becomes more and more difficult to improve. In recent years, low precision computing that uses FP32 or FP16 has attracted much attention. And some new hardware with specific units for low precision computing appears, and some of them show the possibility of strong performance in low precision computing. Additionally, low precision computing has already been accepted and used in some new practical applications, such as AI, machine learning, high performance computing.

There is no doubt that low precision computing (e.g., FP32 or FP16) has advantages compared with standard precision computing (e.g., FP64). One is the memory cost for storage. Take FP32 as an example. It uses 32 bits to store data, which requires less memory than FP64, thus reducing the cost of data storage. Besides, less memory footprint allows better cache memory utilization and improves performance, especially for environments with limited memory resources. The other is in terms of computations. Computations in FP32 format require less data transfer costs than those in FP64 and can accelerate the computation process. Also, the number of elements that can be executed in a single instruction is increased, which can also improve the performance. These advantages will be even greater when programs are performed using hardware suitable for low precision computing.

In many practical applications, a great amount of data and complex computational tasks need to be dealt with. The computation results also have a standard precision requirement. In order to ensure output accuracy, the computations are generally carried out by standard precision computing. Therefore, although it is true that low precision computing can bring a lot of benefits, it is not suitable to simply replace all data and computations with low precision computing, which may lead to the final results failing to meet the application requirements.

Under these situations, it is crucial to determine how to use low precision computing appropriately while ensuring that the output accuracy meets the requirements of practical applications. One possible solution to this issue is to develop new algorithms that use low precision computing and provide results that meet the accuracy requirements (see Figure 1.1). The key idea here is mixed precision(MP) computing [1, 29], which effectively combines both standard precision computing and low precision computing. In this case, standard precision computing is used for critical computations to ensure the accuracy of inputs and outputs, and low precision computing is used for computations that do not strictly require high accuracy during problem-solving. By using such a mixed precision algorithm, we can use and benefit from low precision computing while providing the final results with the same accuracy as traditional algorithms using only standard precision.

In this research, the target problem is a system of linear equations $Ax = b$ in which the coefficient matrix is large, sparse, and non-symmetric. For this problem, iterative refinement and Krylov subspace methods are commonly used in numerical linear algebra. Iterative refinement(IR) [36, 50, 17, 28], proposed by James H. Wilkinson, is an iterative method for improving the accuracy of numerical solutions for linear systems. It is worth noting that iterative refinement is suitable for mixed precision computing. As for Krylov subspace methods [8, 58, 66], they are a class of iterative methods that solve systems of linear equations. For example, the conjugate gradient(CG) method is a typical algorithm for solving symmetric linear systems. And the Generalized Minimal RESidual(GMRES) method [57] and its restart variant GMRES(m) method, as well as the BiConjugate Gradient STABilized(BiCGSTAB) method [65, 66], are often used to solve non-symmetric linear systems. Some studies on mixed precision computing for Krylov subspace methods have been studied [12, 13, 31,

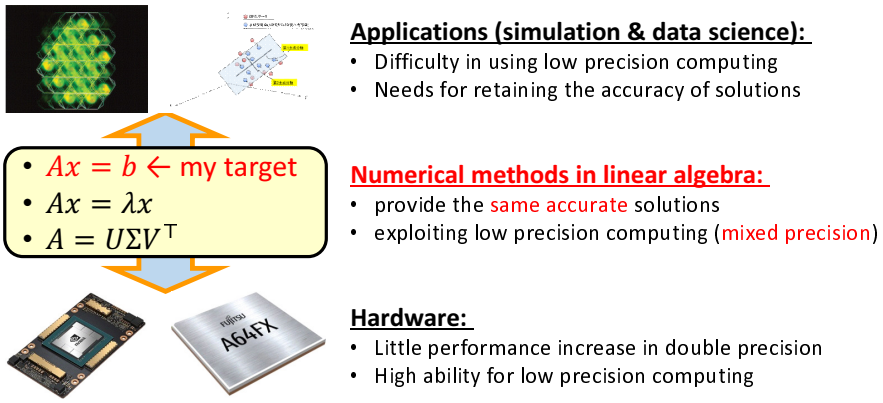


Figure 1.1: Overview of my research.

30, 3, 4, 44, 45, 46, 21, 2, 69].

I consider two Krylov subspace methods that are commonly used to solve the non-symmetric linear system for the target problem: the GMRES(m) and BiCGSTAB methods. Based on the structure of mixed precision iterative refinement(MP-IR), I develop numerical methods using mixed precision computing, which combines FP64 and FP32, and study their numerical characteristic in depth [73, 72]. For the GMRES(m) method, since the traditional GMRES(m) method has the structure of IR [34], its mixed precision variant, which we called MP-GMRES(m), can be reasonably regarded as MP-IR using GMRES(m). And the MP-GMRES(m) method has already been reported and studied [64, 3, 4, 44, 45, 73].

1.2 Research Purpose and Contributions

1.2.1 Research Purpose

The purpose of my research is to develop efficient mixed precision algorithms for solving a linear system $Ax = b$, where A is large, sparse, and non-symmetric. By exploiting the high ability of low precision computing, I aim to gain insights into the new research direction in numerical linear algebra and high performance computing(HPC) and provide new options for accelerating practical applications that need linear solvers. Specifically, research will be conducted from the following two aspects.

- Develop mixed precision algorithms based on iterative refinement and implement them using FP64 and FP32 on a standard CPU platform.
- Conduct comprehensive numerical experiments to investigate the characteristics of the mixed precision algorithms, including numerical behavior

(e.g., accuracy of computed results) and performance (e.g., computational time).

1.2.2 Research Contribution

Mixed precision computing is one of the most recent topics in numerical linear algebra and HPC, and only a few cases studied have been reported. My research contribution lies not only in introducing low precision computing into algorithms but also in finding the algorithms suitable for mixed precision. And there are three main contributions.

- First, I investigate the detailed characteristics and effectiveness of MP-GMRES(m). By conducting comprehensive experiments, I clarify what class of problem can be solved by MP-GMRES(m) and investigate its numerical behavior and performance. A detailed comparison between MP-GMRES(m) and GMRES(m) is also made.
- Another contribution is to propose the MP-IR using BiCGSTAB method and investigate its characteristics and effectiveness. There are few reports on using low precision BiCGSTAB in MP-IR; only an attempt in an application [15]. In my research, I describe how to introduce low precision BiCGSTAB into MP-IR and conduct numerical experiments to evaluate its characteristics and effectiveness.
- Thirdly, my research shows the applicability of mixed precision computing in numerical linear solvers, and its main impact will be accelerating applications that need linear solvers. Besides, reducing the time of the simulation program will also significantly impact data science applications because simulation is now one of the important data generators.

1.3 Structure of the thesis

This thesis consists of five chapters. The brief descriptions of other chapters are as follows.

- **Chapter 2 Related work**

  This chapter introduces mixed precision computing, Krylov subspace methods, and the recent research background of applying mixed precision computing in Krylov subspace methods, including their definitions, recent related research, and applications.

- **Chapter 3 Numerical Investigation of The Mixed Precision GMRES(m) Method**

  In this chapter, the GMRES(m) method and its mixed precision variant, MP-GMRES(m) method, are studied, including the introduction of methods, theoretical analysis, detailed experiment results, and comparison with

traditional GMRES(m). Detailed analysis and comparison are given from three aspects: the maximum attainable accuracy, the number of iterations, and the execution time.

- **Chapter 4 Numerical Investigation of The Mixed Precision Iterative Refinement using the BiCGSTAB Method**

  The research on MP-IR using BiCGSTAB is presented in this chapter. First, the BiCGSTAB method is introduced. Then, how the BiCGSTAB method is employed in a mixed precision variant of IR is explained. Finally, detailed numerical experiment results and their analysis and comparison with traditional BiCGSTAB, as well as MP-GMRES(m), are provided.

- **Chapter 5 Conclusion and Future Work**

  In this chapter, an overall summary and contributions of the research are given, as well as future work.

Chapter 2

Related Work

2.1 Mixed Precision Computing

Mixed precision computing is the technique that combines different precisions and achieves the same computational accuracy as using standard precision computing. By mixed precision computing, we can reduce the cost of data storage and transfer in computations, as well as increase the arithmetic performance. In mixed precision computing, the commonly used precision includes standard precision (generally FP64, also called high precision in this thesis), single precision (generally FP32), half precision (generally FP16), etc. The formats of the above three floating points [23] are shown in Figure 2.1. The core idea is to use standard precision, such as FP64, for some parts of the computation to maintain numerical stability and the computation results accuracy while using lower precision, e.g., FP32 or FP16, for others to improve the computation efficiency.

Mixed precision computing has a wide range of applications in scientific computation, numerical linear algebra, AI, deep learning, and other fields. It is often used to accelerate the model training process and computation process, etc., and its effectiveness has been confirmed in many practical applications [48, 39, 55]. In recent years, with the appearance and development of some hardware suitable for low precision computing, such as NVIDIA GPU with Tensor Cores, its effectiveness is expected to be further improved.

In the field of scientific computing and numerical linear algebra, J. Sun et al. used mixed precision computing to improve the performance of direct solvers on FPGAs [62]. They proposed a new architecture for a mixed precision solver and confirmed its effectiveness on FPGAs by analyzing the performance of different precisions and mixed precision iterative refinement. Marc Baboulin et al. [5] gave the derivation of mixed precision variants for direct and iterative methods, and experiments showed that mixed precision algorithms could enhance performance when solving linear systems using dense/sparse direct methods and sparse iterative methods. Hiroyuki Ootomo and Rio Yokota [54] study a matrix-matrix multiplication implementation using FP16 and FP32 Tensor

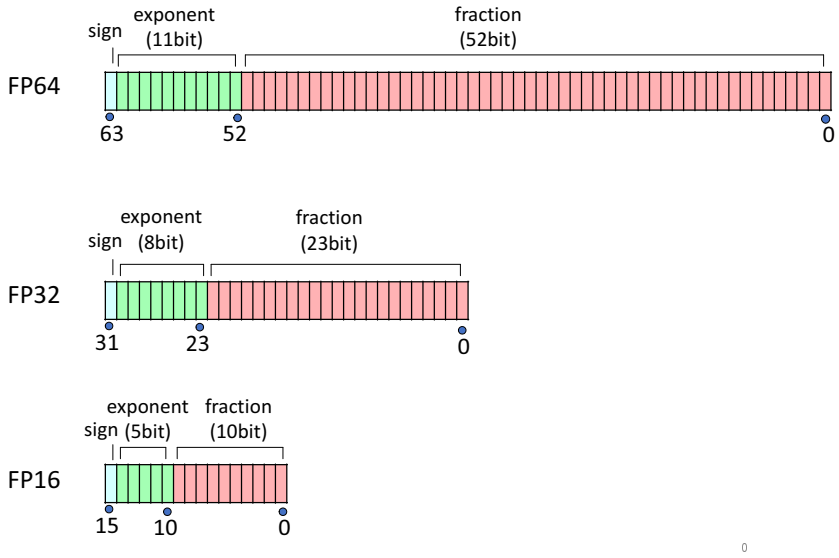


Figure 2.1: The format of standard precision(FP64), single precision(FP32), and half precision(FP16).

Cores, respectively, and achieve excellent throughput. Iterative refinement is a commonly used method in numerical linear algebra, which iteratively improves the approximate solution. And mixed precision computing is often combined with this method, which is shown in Figure 2.2. The basic idea behind it is to compute the correction term by low precision computing (e.g., single or half precision), and its solution is updated and checked by double precision. More related works of mixed precision computing in these fields can be found in Section 2.3.

In addition to the field of scientific computation and numerical linear algebra, the effectiveness of mixed precision computing has also been demonstrated in other fields as well.

In the field of natural language processing(NLP), mixed precision computing techniques can be used to accelerate the training process of large language models. Oleksii Kuchaiev et al. introduced a toolkit based on Tensorflow featuring mixed precision that can greatly accelerate the training process of neural networks and illustrated its effectiveness through some tasks on machine translation and speech recognition [39]. Houwen Peng et al. presented a new FP8 mixed precision framework to train large language models and demonstrated that this new framework can reduce memory and speed up operation [55].

Besides, T. Ichimura et al. successfully applied mixed precision computing to earthquake simulation and achieved a significant speedup by combining double, single, and half precision [32]. T. Kurth et al. used FP16 in NVIDIA V100 Tensor Core GPUs to identify extreme weather in climate analysis [41].

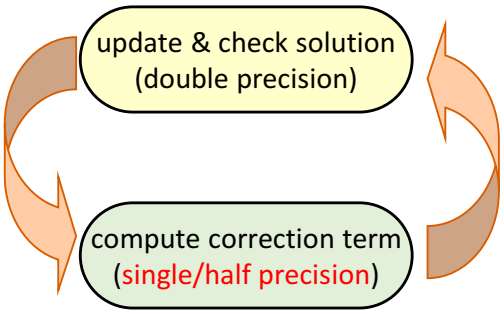


Figure 2.2: Mixed precision computing based on iterative refinement.

Mixed precision technique is also one of the popular topics in the AI and deep learning field. For example, the technique is used to accelerate training models such as deep neural networks. Some related reports can be found [24, 47, 48]. NVIDIA has given a report on Mixed precision training with a brief introduction of three effective mixed precision training techniques [47]. Detailed description and results were reported in [48]. In [48], Paulius Micikevicius et al. described a technique for training deep neural networks (DNNs) using half precision and demonstrated that the technique is effective in reducing memory consumption and works well for a wide range of deep network models.

2.2 Krylov Subspace Methods

The Krylov subspace method, listed as one of the Top 10 Algorithms of the 20th century [14], is one of the important research topics in the field of numerical linear algebra and scientific computation. The subspace was first introduced by Russian applied mathematician and naval engineer Alexei Krylov in 1931 [38], and its applications to iterative methods, that is, the Krylov subspace method, gradually appeared. The Krylov subspace methods play a vital role in science and engineering and are commonly used to solve a large and sparse linear system $Ax = b$, where A is a large and sparse coefficient matrix. The basic idea behind these methods is to find an approximate solution which satisfies the accuracy requirement in a Krylov subspace with a smaller dimension.

For a general linear system $Ax = b$, let x_0 be an initial guess, and a k-dimensional Krylov subspace can be defined as

$$\mathscr{K}_k(A, r_0) = \text{span}\left\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\right\}, \tag{2.1}$$

where $r_0 = b - Ax_0$. In general, We can find the k-th approximate solution x_k as

$$x_k = x_0 + z_k, \quad z_k \in \mathscr{K}_k(A, r_0). \tag{2.2}$$

When choosing z_k, different conditions derive different methods. For example, the Conjugate Gradient (CG) method [27] is derived from the condition

$$r_k \perp \mathscr{K}_k(A, r_0), \tag{2.3}$$

where $r_k = b - Ax_k$, while the Generalized Minimum RESidual (GMRES) method is derived from the condition

$$r_k \perp A\mathscr{K}_k(A, r_0). \tag{2.4}$$

Other common Krylov subspace methods include the Biconjugate Gradient Stabilized (BiCGSTAB) method, the Minimal Residual (MINRES) method, and so on.

Studies on the Krylov subspace method and its variants have been reported. In [8, 19, 18], some chapters give introductions to several Krylov subspace methods such as CG, MINRES, BiCG, BiCGSTAB, QMR, including their derivation, implementation, and some optimizations. A summary of known convergence results for the CG, MINRES, and GMRES methods is summarized and provided by Liesen, J. et al. [43]. The convergence bounds of each of these three methods, such as residual norm, are discussed one by one. Saad Youcef provided a study on implementing Krylov subspace methods on vector and parallel computers [56]. Besides, Sleijpen, G. L. et al. confirmed the multiple implementations of BiCGSTAB(l), which allow the combinations of BiCG with arbitrary polynomial methods [60]. L. T. Yang et al. presented an improved variant of the BiCGSTAB method, which is called the IBiCGStab. This variant avoids the increase in computational cost while maintaining the advantages of the original method [70].

Preconditioning and parallelization techniques are good choices for improving the performance of Krylov subspace methods, aiming to accelerate the iterative convergence process and reduce the computational complexity of solving linear systems. R. Barrett et al. provided a detailed description of preconditioning in iterative methods, including why and how to preconditioning, as well as some typical preconditioning such as Jacobi preconditioning, symmetric successive over-relaxation (SSOR) preconditioning, Incomplete Factorization preconditioners (e.g., ILU(0)) and some others [8]. An effective preconditioning technique can enhance the effectiveness of Krylov subspace methods in practical applications [22]. R. Barrett et al. [8] introduced preconditioning and gave some typical preconditioners. Several studies showed the effectiveness of preconditioners, e.g., ILU and the preconditioner based on IGO (incomplete Givens orthogonalization), in subspace methods such as GMRES method [49, 6]. Some preconditioners are also reported in the studies of application in BiCGSTAB, including some applications for solving practical problems [59, 61]. In [59], the Scheduled Relaxation Jacobi (SRJ) method is applied as a preconditioner in BiCGSTAB method. Other preconditioners, such as Sparse Approximate Inverses, can also be found [37].

High-performance parallelization also plays a great role in the optimization of Krylov subspace methods. Vuik, C. et al. described a parallel implementation of GMRES using ILU preconditioner for the discretized incompressible Navier-Stokes equations problem [67]. Xu applied OpenMP parallel into ILU(0) GMRES, which occupies most of the computations, and improved the performance of simulations on incompressible fluid flows [68].

Through the effective use of preconditioning and parallelization techniques, we are able to utilize the computational resources better, thus improving the solution efficiency in solving large-scale problems.

2.3 Studies on Mixed Precision Computing for Krylov Subspace Methods

In recent years, mixed precision computing has attracted much attention in the field of numerical linear algebra, as well as the field of high-performance computing. Abdelfattah et al. [1] and Higham et al. [29] provided some reports on mixed precision computing, which include a review and discussion of the latest trends in computational hardware.

In numerical linear algebra, mixed precision computing techniques, including the combination of standard precision (generally FP64) and higher precision, are often used in conjunction with iterative refinement (IR), which means the mixed precision variant of iterative refinement. This mixed precision variant plays an important role in solving linear systems and contributes to improving the accuracy of the solution, especially for ill-conditioned problems [17, 28]. Additionally, similar cases are also reported in the problems of eigenvalue, inverse LU, and QR factorization [51, 52, 53].

As low precision computing continues evolving, its high ability has attracted widespread attention. In the late 2000s, the wide application of GPGPU promoted the further development of mixed precision computing. Mixed precision computing that combines both standard and low precision computing attracted much attention and in-depth research and was applied to the field of numerical linear algebra to achieve high performance and accelerate practical problems.

In dealing with dense linear systems, early works and a summary are provided [42, 11, 5]. As for sparse linear systems, the performance of the iterative refinement using the sparse LU factorization in FP32 was presented [10]. In addition, more studies, including the efficiency of the Krylov subspace method using a preconditioner in FP32, have also attracted attention [10, 63, 33]. Similar attempts are also made on GPU [25].

Some theoretical studies are also reported. The GMRES-IR, proposed by Carson and Higham, is obtained based on a new rounding error analysis [12]. They also analyzed the GMRES-IR method using three precisions (FP64, FP32, FP16) from the theoretical aspect [13]. In addition, this method was subsequently extended to general linear systems [31] and symmetric positive definite problems [30].

The introduction of NVIDIA GPU with Tensor cores has further advanced the development of low precision computing, including half precision, which is expected to promote more research on mixed-precision methods. For example, Haidar et al. provided the performance using FP16 provided by Tensor Cores [26]. Error analysis of FMA operations using mixed precision blocks provided by Tensor Cores is also presented [9]. Moreover, a new benchmark,

called HPL-AI, is proposed. This benchmark is used to evaluate the ability of low precision computing of supercomputer systems and has been executed on many supercomputers, e.g., Summit, Fugaku [40].

Turner and Walker provided the early works of the mixed precision variant of the GMRES(m) method using FP64 and FP32 [64]. Its performance on early GPU was reported by Anzt et al. [3, 4]. Detailed discussions on mixed precision GMRES(m) method with or without a preconditioner on both CPU and GPU platforms were made [44, 45, 46]. As for the BiCGSTAB method, low precision computing was attempted to be applied in a QCD problem [15].

In addition to the above studies on GMRES(m) and BiCGSTAB, other subspace methods are also studied. For instance, some attempts to employ low precision computing into the CG method were made [21, 2, 69].

The studies mentioned above mainly focus on successful results and practical applications. Compared with these studies, our study has two main contributions. One is the detailed discussion on numerical characteristics of mixed precision algorithms from the viewpoints of attainable accuracy, execution time, etc. The other is proposing a mixed precision algorithm based on mixed precision iterative refinement and BiCGSTAB. Detailed analysis and comparison are also given in this study. Our study helps provide a clearer and more comprehensive understanding of the performance of mixed precision computing in Krylov subspace methods and provides some new sights in applications.

Chapter 3

Numerical Investigation of The Mixed Precision GMRES(m) Method

In this chapter, the GMRES(m) method and its mixed precision variant are investigated. First, a general introduction to this part of the research is given. Then, iterative refinement, the GMRES(m) method, and their mixed precision variants are introduced. Next, the methods are analyzed from the theoretical aspect. Finally, detailed analysis and comparison are given.

3.1 Introduction

The GMRES(m) method is one of the well-known Krylov methods, which is a restart variant of the GMRES method. It is often used to solve problems of sparse and non-symmetric linear systems. Currently, mixed precision computing has been successfully applied to the GMRES(m) method and has been studied by many scholars [64, 34, 3, 4, 44, 45, 46]. From these studies and reports, there is strong evidence that shows the effectiveness of mixed precision computing for some test problems or applications. For some problems, using the mixed precision GMRES(m) solver can also satisfy the accuracy requirement and even show a shorter execution time on the CPU or GPU platforms, which helps improve performance. However, many studies and reports are conducted for specified applications. The numerical behavior of the mixed precision GMRES(m) method and the class of problems that can be solved have not been clarified in detail.

The research on the mixed precision GMRES(m) method aims to investigate the numerical behavior from the viewpoints of, for example, attainable accuracy, execution time, etc. With multiple sets of parameter settings, comparative experiments are conducted for various test problems (matrices) to study the mixed precision GMRES(m) method in depth and compare it with the traditional

GMRES(m) method using only FP64. Besides, the convergence process of the algorithm is also one of the research interests, which helps us better understand the algorithm process and the experiment results. Thus, for some typical cases, the histories of the relative residual 2-norm are investigated and analyzed. The presented results and conclusions will contribute to a deeper understanding and effective application of the method.

3.2 Iterative Refinement and its Mixed Precision Variant

Iterative refinement(IR) [36, 50, 17, 28], proposed by James H. Wilkinson, is a long-standing method, and the solution x can be iteratively improved and reach the required accuracy by it. The scheme consists of the following three steps: Let $\tilde{x}$ be the current approximate solution, then

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

These three steps are repeated. If the convergence criterion is satisfied, the iteration is stopped; otherwise, return to Step 1 and perform more iterations using the latest approximate solution until convergence.

In Step 2, different numerical methods are selected depending on the characteristics of the error equation. If the approximate solution $\tilde{x}$ can be improved in Step 2, like

$$\|b - A(\tilde{x} + \tilde{e})\|_2 < \|b - A\tilde{x}\|_2, \tag{3.1}$$

we can eventually obtain a sufficiently accurate solution.

Iterative refinement is very suitable for mixed precision computing, and we can introduce low precision computing to it. In mixed precision iterative refinement(MP-IR), Step 1 and Step 3 are performed with high precision (or standard precision), and Step 2 is performed with low precision. The choice of precision is determined by their influence on the final accuracy. For Step 1, if the residual is computed in low precision, the accuracy will limit the subsequent error correction. Also, the approximate solution needs to be updated in high precision to avoid round-off caused by low precision and to perform convergence checks.

In contrast, the computation in Step 2 does not strictly require high precision computing. First, it is known that the solution can potentially achieve the same accuracy that high precision computing does for some certain problems. And second, reducing the computational accuracy in this step does not limit the maximum attainable accuracy. If the approximate solution is not accurate enough in this iteration, more iterations can be performed until the convergence criterion is satisfied.

Due to these reasons, we can obtain the mixed precision iterative refinement, whose details are shown in Algorithm 1.

Algorithm 1 Mixed Precision Iterative Refinement (MP-IR)

Input: $x(= x_0)$: initial guess, A: coefficient matrix, b: right-hand side vector, ϵ: convergence criterion

1: $A^{(\mathrm{L})} = \mathrm{Low}(A)$ ▷ convert A to low precision data
2: **repeat**
3: $r = b - Ax$
4: **if** $\|r\|_2/\|b\|_2 \leq \epsilon$ **then return** x
5: $r^{(\mathrm{L})} = \mathrm{Low}(r)$ ▷ convert r to low precision data
6: Solve $A^{(\mathrm{L})}e^{(\mathrm{L})} = r^{(\mathrm{L})}$ ▷ solve by *low precision* arithmetic
7: $e = \mathrm{Std}(e^{(\mathrm{L})})$ ▷ convert $e^{(\mathrm{L})}$ to standard precision data
8: $x = x + e$
9: **until** attain other conditions (e.g., limit of the total iterations)

Output: x: (approximate) solution vector

In the algorithm, variables with the suffix "(L)" are stored in low precision, and Low($\cdot$) and Std($\cdot$) represent the operation of converting a variable from standard precision to low precision and from low precision to standard precision, respectively. The accuracy of the returned solution is checked by standard precision (in Line 4), which uses relative residual 2-norm for the check. In this research, FP64 and FP32 are used as standard precision and low precision, respectively.

3.3 Review of GMRES(m) and Mixed Precision GMRES(m) Method

3.3.1 The GMRES(m) Method

The Generalized Minimal RESidual(GMRES) method [58] is an iterative method for solving systems of linear equations with a large, sparse, and non-symmetric coefficient matrix. The method was proposed by Saad and Schultz [57] in 1986 and is one of the famous Krylov subspace methods. In the case of setting the initial guess $x_0 = 0$, the key idea is to construct a Krylov subspace and find an approximate solution in that subspace. In this research, the target problem is the system of linear equations

$$Ax = b, \tag{3.2}$$

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 (3.2), which is obtained at the k-th iteration of the GMRES method. The GMRES method consists of the following steps.

Step 1: Construct the Krylov subspace.

Give an initial guess x_0 and compute residual $r_0 = b - Ax_0$, and construct the Krylov subspace $\mathscr{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$. GMRES method can be described as finding an approximate solution $x_k \in x_0 + \mathscr{K}_k(A, r_0)$ that satisfies the condition $r_k \perp A\mathscr{K}_k(A, r_0)$, where $r_k = b - Ax_k$.

Step 2: Obtain an orthogonal matrix V_k and Hessenberg matrix $\tilde{H}_k$ using Arnoldi process.

The Arnoldi process is the process of computing a set of orthogonal bases of a Krylov subspace. In the GMRES method, this process is used to project and orthogonalize the coefficient matrix A into the Krylov subspace, which involves the modified Gram-Schmidt orthogonalization (MGS) and sparse matrix vector multiplication (SpMV).

Through Arnoldi process, an orthogonal 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 $(k+1)$-by-k upper Hessenberg matrix

$$\tilde{H}_k = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} \\ & h_{3,2} & \cdots & h_{3,k} \\ & & \ddots & \vdots \\ & & & h_{k+1,k} \end{pmatrix} \in \mathbb{R}^{(k+1)\times k} \tag{3.3}$$

are obtained. The relation between V_k and $\tilde{H}_k$ is

$$AV_k = V_{k+1}\tilde{H}_k. \tag{3.4}$$

These two matrices are used to approximately represent the subspace and thus used to solve the subsequent least squares problem.

Step 3: Solve least squares problem

After step 2, the approximate solution can be written as

$$x_k = x_0 + V_k y_k, \tag{3.5}$$

and the problem reduces to finding a vector y_k, which minimizes the 2-norm of the residual $r_k = b - Ax_k$. In other words, that is solving the least squares problem

$$y_k = \arg\min_{y\in\mathbb{R}^k} \|r_k\|_2$$

Using (3.5), we have

$$\begin{aligned} y_k &= \arg\min_{y\in\mathbb{R}^k} \|r_k\|_2 \\ &= \arg\min_{y\in\mathbb{R}^k} \|b - A(x_0 + V_k y)\|_2 \\ &= \arg\min_{y\in\mathbb{R}^k} \|r_0 - AV_k y\|_2. \end{aligned} \tag{3.6}$$

By the relation (3.4), the minimization problem (3.6) can be transformed as follows:

$$\begin{aligned}\min_{y\in\mathbb{R}^k}\|r_0 - AV_k y\|_2 &\Leftrightarrow \min_{y\in\mathbb{R}^k}\|r_0 - V_{k+1}\tilde{H}_k y\|_2 \\ &\Leftrightarrow \min_{y\in\mathbb{R}^k}\|V_{k+1}^\top r_0 - \tilde{H}_k y\|_2 \\ &\Leftrightarrow \min_{y\in\mathbb{R}^k}\|\beta u - \tilde{H}_k y\|_2, \end{aligned} \tag{3.7}$$

where $\beta = \|r_0\|_2$ and $u = [1, 0, \ldots, 0]^\top \in \mathbb{R}^{k+1}$. Then, the minimization problem (3.7) is solved by the QR factorization based on the Givens rotation.

Finally, the approximate solution can be obtained.

The GMRES method is suitable for large, sparse, and non-symmetric linear systems and has good convergence. By iterative computations, the solution can be gradually improved.it can obtain a solution that approximates the exact solution. However, as the number of iterations increases, storage and computation costs also increase at each iteration step. Thus, when the number of iterations is quite large and reaches the memory limitation, the GMRES method may gradually lose its competitiveness. Due to this fact, some variants of the GMRES method have been explored, and the restart GMRES method [58], which is known as the GMRES(m) method, is one of them.

The outline of the GMRES(m) method is shown in Algorithm 2. In GMRES(m), m represents the restart frequency, and the method consists of the following two main steps:

- **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: $x_0 = x_m$.

These two steps are repeated until the algorithm converges.

As seen from Line 6 in Algorithm 2, GMRES(m) includes m-step Arnoldi process, which is also an iterative process. Therefore, it has a nested loop structure. For a better description, we use *inner iteration* to represent the number of iterations of m-step Arnoldi process in GMRES(m) method.

Compared with the GMRES method, the GMRES(m) method uses a restart strategy to keep the number of iterations and the dimension of the subspace within an acceptable range (e.g., restart frequency m), which is helpful in reducing the storage and computation cost and may improve the performance for some problems. For the choice of restart frequency, the discussion will be given in later sections.

3.3.2 The Mixed Precision GMRES(m) Method

In the standard GMRES(m) method, all computations are in high precision. Thus, which part of the computations is suitable for low precision computing is

Algorithm 2 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** $\beta/\|b\|_2 \leq \epsilon$ **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

necessary to be identified to derive a mixed precision variant of the GMRES(m) method.

As mentioned in the section 3.3.1, the GMRES(m) method has a similar loop structure to IR. And the relation between GMRES(m) and IR has been pointed out by Imakura et al. [34], that is, the following computation in GMRES(m):

- **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 computations:

- **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 equivalent transformation suggests that the GMRES(m) method can be reasonably viewed as an iterative refinement that solves the error equation by the m-iteration GMRES method. Thus, based on the structure of MP-IR, a mixed precision variant of iterative refinement using m-iteration GMRES method can be derived, which can also be regarded as the mixed precision variant of GMRES(m) method.

The mixed precision GMRES(m) method can be described as following steps:

- **Step 1:** Compute the residual $r = b - A\tilde{x}$ in *high precision*.
- **Step 2:** Solve the error equation $Ae = r$ by the m-iteration GMRES and obtain an approximate solution $\tilde{e}$ in *low precision*.
- **Step 3:** Update the solution $\tilde{x} = \tilde{x} + \tilde{e}$ in *high precision*.

And Algorithm 3 shows the outline of the mixed precision GMRES(m) method. Similar as Algorithm 1, the suffix "(L)" means that variables are stored in low precision, and Low($\cdot$) and Std($\cdot$) represent the operation of converting a variable from standard precision to low precision and from low precision to standard precision, respectively.

Algorithm 3 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

3.4 Theoretical Analysis of Mixed Precision GMRES(m) Method

In this section, the mixed precision GMRES(m) using FP64 and FP32 is derived from the theoretical aspect, including the details of the method, analysis of numerical behavior, estimation of the speedup, and memory cost. Then, it is compared with the traditional GMRES(m) method using only FP64. For simplicity in description, we denote the traditional GMRES(m) method by "FP64-GMRES(m)" and mixed precison GMRES(m) by "MP-GMRES(m)".

3.4.1 Analysis on the Numerical Behavior

The MP-GMRES(m) method has already been proposed, and some related research can be found [64, 3, 4, 44, 45]. However, the numerical behavior of

the method is not clear, which is also one of my research objectives. Here, Algorithm 4 presents the details of the MP-GMRES(m) method and some analysis of the numerical behavior from the theoretical aspect is given.

In Algorithm 4, variables with the suffix "(FP32)" are stored in FP32, and lines with the comment "by FP32" are computed by the FP32 arithmetic. k_{total} counts the total number of inner iterations.

Next, the numerical behavior of the MP-GMRES(m) method is analyzed following the steps of MP-GMRES(m) and compared with the FP64-GMRES(m) method.

- **Step 1:** Compute residual in *high precision*.

  In Line 7, the residual r is obtained using A and b stored in FP64. And the residual itself is also stored in FP64. This step is in the same condition as FP64-GMRES(m).

- **Step 2:** Solve error equation in *low precision*

  In the m-iteration GMRES computation, that is lines 11-33, low precision is used for all variables and computations. However, owing to the comparatively limited computational capabilities of low precision computing, the MP-GMRES(m) method is expected to exhibit a lower refinement ratio of the approximate solution for the same number of iterations(i.e., with the same restart frequency m). Therefore, MP-GMRES(m) may require more outer iterations(line 6-37) to achieve convergence.

  However, the MP-GMRES(m) method is able to converge as long as the approximate solution can be improved in each iteration

  $$\|b - Ax_k\|_2 < \|b - Ax_0\|_2, \tag{3.8}$$

  even though sometimes the improvement is only a little.

- **Step 3:** Update approximate solution in *high precision*

  Both the approximate solution updating and convergence check are performed in FP64, which is the same as the FP64-GMRES(m) method. Therefore, the result is as accurate as those obtained by the FP64-GMRES(m) method if the MP-GMRES(m) method finally converges.

  In addition, a new phenomenon may occur in this step. In Step 2, the relative residual 2-norm is also checked in each inner iteration (Line 30). When the convergence criterion is satisfied, it jumps out of the loop and goes directly to Step 3. The problem, however, is that the computation and check in Step 2 are performed in FP32. As a result, there may be a situation where the approximate solution that passes the inner convergence check (Line 30) does not satisfy the outer convergence criterion (Line 9). In this case, more iterations are required, which also leads to an increase in the total number of iterations.

In addition, the MP-GMRES(m) method differs from the FP64-GMRES(m) method in terms of the subspace and the requirement of restart frequency.

Algorithm 4 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

$A^{(\mathrm{FP32})} = \mathrm{ToFP32}(A)$
$\zeta = \|b\|_2$
$\zeta^{(\mathrm{FP32})} = \mathrm{ToFP32}(\zeta)$
$\epsilon^{(\mathrm{FP32})} = \mathrm{ToFP32}(\epsilon)$
$k_{\mathrm{total}} = 0$
repeat
  $r_0 = b - Ax_0$
  $\beta = \|r_0\|_2$
  if $\beta/\zeta \leq \epsilon$ **then return** x_0
  $v_1 = r_0/\beta$
  $\beta^{(\mathrm{FP32})} = \mathrm{ToFP32}(\beta)$
  $v_1^{(\mathrm{FP32})} = \mathrm{ToFP32}(v_1)$
  $u^{(\mathrm{FP32})} = [u_1^{(\mathrm{FP32})}, \ldots, u_{k+1}^{(\mathrm{FP32})}]^\top = [\beta^{(\mathrm{FP32})}, 0, \ldots, 0]^\top$
  for $k = 1, 2, \ldots, m$ **do**
    $k_{\mathrm{total}} = k_{\mathrm{total}} + 1$
    $w^{(\mathrm{FP32})} = A^{(\mathrm{FP32})} v_k^{(\mathrm{FP32})}$ ▷ by FP32
    for $j = 1, 2, \ldots, k$ **do**
      $h_{j,k}^{(\mathrm{FP32})} = {v_j^{(\mathrm{FP32})}}^\top w^{(\mathrm{FP32})}$ ▷ by FP32
      $w^{(\mathrm{FP32})} = w^{(\mathrm{FP32})} - h_{j,k}^{(\mathrm{FP32})} v_j^{(\mathrm{FP32})}$ ▷ by FP32
    end for
    $h_{k+1,k}^{(\mathrm{FP32})} = \|w^{(\mathrm{FP32})}\|_2$ ▷ by FP32
    $v_{k+1}^{(\mathrm{FP32})} = w^{(\mathrm{FP32})}/h_{k+1,k}^{(\mathrm{FP32})}$ ▷ by FP32
    for $j = 1, 2, \ldots, k-1$ **do**
      $\begin{pmatrix} h_{j,k}^{(\mathrm{FP32})} \\ h_{j+1,k}^{(\mathrm{FP32})} \end{pmatrix} = \begin{pmatrix} c_j^{(\mathrm{FP32})} & s_j^{(\mathrm{FP32})} \\ -s_j^{(\mathrm{FP32})} & c_j^{(\mathrm{FP32})} \end{pmatrix} \begin{pmatrix} h_{j,k}^{(\mathrm{FP32})} \\ h_{j+1,k}^{(\mathrm{FP32})} \end{pmatrix}$ ▷ by FP32
    end for
    $c_k^{(\mathrm{FP32})} = h_{k,k}^{(\mathrm{FP32})} / \sqrt{{h_{k,k}^{(\mathrm{FP32})}}^2 + {h_{k+1,k}^{(\mathrm{FP32})}}^2}$ ▷ by FP32
    $s_k^{(\mathrm{FP32})} = h_{k+1,k}^{(\mathrm{FP32})} / \sqrt{{h_{k,k}^{(\mathrm{FP32})}}^2 + {h_{k+1,k}^{(\mathrm{FP32})}}^2}$ ▷ by FP32
    $\begin{pmatrix} u_k^{(\mathrm{FP32})} \\ u_{k+1}^{(\mathrm{FP32})} \end{pmatrix} = \begin{pmatrix} c_k^{(\mathrm{FP32})} & s_k^{(\mathrm{FP32})} \\ -s_k^{(\mathrm{FP32})} & c_k^{(\mathrm{FP32})} \end{pmatrix} \begin{pmatrix} u_k^{(\mathrm{FP32})} \\ u_{k+1}^{(\mathrm{FP32})} \end{pmatrix}$ ▷ by FP32
    $\begin{pmatrix} h_{k,k}^{(\mathrm{FP32})} \\ h_{k+1,k}^{(\mathrm{FP32})} \end{pmatrix} = \begin{pmatrix} c_k^{(\mathrm{FP32})} & s_k^{(\mathrm{FP32})} \\ -s_k^{(\mathrm{FP32})} & c_k^{(\mathrm{FP32})} \end{pmatrix} \begin{pmatrix} h_{k,k}^{(\mathrm{FP32})} \\ h_{k+1,k}^{(\mathrm{FP32})} \end{pmatrix}$ ▷ by FP32
    if $|u_{k+1}^{(\mathrm{FP32})}|/\zeta^{(\mathrm{FP32})} \leq \epsilon^{(\mathrm{FP32})}$ **then goto** 32 ▷ by FP32
  end for
  $y_k^{(\mathrm{FP32})} = \begin{pmatrix} h_{1,1}^{(\mathrm{FP32})} & \cdots & h_{1,k}^{(\mathrm{FP32})} \\ & \ddots & \vdots \\ & & h_{k,k}^{(\mathrm{FP32})} \end{pmatrix}^{-1} \begin{pmatrix} u_1^{(\mathrm{FP32})} \\ \vdots \\ u_k^{(\mathrm{FP32})} \end{pmatrix}$ ▷ by FP32
  $z_k^{(\mathrm{FP32})} = \begin{bmatrix} v_1^{(\mathrm{FP32})} & \cdots & v_k^{(\mathrm{FP32})} \end{bmatrix} y_k^{(\mathrm{FP32})}$ ▷ by FP32
  $z_k = \mathrm{ToFP64}(z_k^{(\mathrm{FP32})})$
  $x_k = x_0 + z_k$
  $x_0 = x_k$
until $k_{\mathrm{total}} > k_{\max}$
Output: x_0

- In the MP-GMRES(m) method, the generated Krylov subspace is $\mathscr{K}_k(A^{(\mathrm{FP32})}, v_1^{(\mathrm{FP32})})$, while $\mathscr{K}_k(A, v_1)$ is generated in FP64-GMRES(m).
- The error equation solved in MP-GMRES(m) method is different from that of FP64-GMRES(m). MP-GMRES(m) solves

$$A^{(\mathrm{FP32})}e = r_0^{(\mathrm{FP32})}(= \beta^{(\mathrm{FP32})}v_1^{(\mathrm{FP32})}), \tag{3.9}$$

  while FP64-GMRES(M) solves

$$Ae = r_0(= \beta v_1). \tag{3.10}$$

  Therefore, the requirement for restart frequency m during the iteration process is also different. In general, MP-GMRES(m) requires a larger m. And the discussion about restart frequency m is given in detail in the subsequent sections based on the experimental results.

3.4.2 Memory Footprint on Standard CPU Platform

On standard CPU platforms, both FP64 and FP32 are available data types, and the conversion of variables between these two types is straightforward. For example, in the C language, one can use `double` and `float` to represent FP64 and FP32, respectively, and type cast can be used to realize the transformation between the two. Thus, the MP-GMRES(m) algorithm can theoretically be implemented on CPU platforms. However, it is crucial to be mindful of the associated memory costs.

Compared to the FP64-GMRES(m) method, the MP-GMRES(m) method needs to convert the data in the coefficient matrix A to low precision and store it in a new array $A^{(\mathrm{FP32})}$(Line 1 in Algorithm 4), which leads to more memory consumption. However, MP-GMRES(m) also benefits from reduced memory consumption. By replacing V_{m+1} with $V_{m+1}^{(\mathrm{FP32})}$, the memory consumption can be reduced. And, V_{m+1} is not required in the MP-GMRES(m) method, further contributing to a positive impact on memory footprint reduction.

In our research, the CSR (Compressed Sparse Row) format is used to store a sparse matrix. The CSR format generally consists of three one-dimensional arrays: `val`, `col_ind` and `row_ptr`. The specific representation is as follows.

- `val`: used to store the values of non-zero elements in the matrix.
- `col_ind`: used to store the column indices of all non-zero elements.
- `row_ptr`: used to store the position indices of the first non-zero element in each row in `val` array.

For the MP-GMRES(m) method, only `val` needs to be stored in FP32 additionally.

The memory footprint of the FP64-GMRES(m) method and MP-GMRES(m) method is compared in Table 3.1, where N_{nz} denotes the number of non-zero

Table 3.1: Comparison of memory footprint (in bytes) between FP64-GMRES(m) and MP-GMRES(m) method.

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

elements in the matrix A. In Table 3.1, the change in memory footprint from FP64-GMRES(m) to MP-GMRES(m) is

$$\begin{aligned}(16N_{\text{nz}} + (4m+8)n) - (12N_{\text{nz}} + (8m+12)n) &= 4N_{\text{nz}} - 4mn - 4n \\ &= 4n\left(\frac{N_{\text{nz}}}{n} - (m+1)\right). \quad (3.11)\end{aligned}$$

This equation shows that if the average of the number of non-zero elements per row (N_{nz}/n) in A is smaller than the restart frequency m, that is

$$\frac{N_{\text{nz}}}{n} < m, \quad (3.12)$$

the memory footprint does not increase. And this condition is often satisfied in practical applications.

3.4.3 Estimation on the Expected Speedup

Besides numerical behavior and memory footprint, another critical aspect is whether the MP-GMRES(m) method obtains improvement in execution time. This subsection discusses the estimated speedup using a simple model.

The model primarily focuses on the m-step Arnoldi process and ignores other computational costs. Also, the analysis and modeling do not consider the impact of the cache memory in order to simplify the comparison of the m-step Arnoldi process in FP64 and FP32.

The m-step Arnoldi process mainly consists of SPMV and MGS operations for m vector, both of which can be assumed as memory-bound computations. Table 3.2 roughly estimates the memory access cost in the m-step Arnoldi process. For SpMV, we consider the memory access cost for two vectors, as well as the cost of storing the matrix in CSR format.

The number of iterations required for the m-iteration GMRES process also differs in FP64-GMRES(m) and MP-GMRES(m) and effects the estimated speedup. Let γ donate the ratio of the number of repetitions of the m-iteration

Table 3.2: Rough estimation of the memory access cost (in bytes) in the m-step Arnoldi process in FP64 and FP32.

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

GMRES process in MP-GMRES(m) over that in FP64-GMRES(m). In general, it is expected to be greater than 1. The acceleration of MP-GMRES(m) compared to FP64-GMRES(m) is estimated as follow

$$\begin{aligned}(\text{Speedup}) &= \frac{4m^2n + 12mN_{\text{nz}} + 20mn}{\gamma(2m^2n + 8mN_{\text{nz}} + 12mn)} \\ &= \frac{2m + 6\frac{N_{\text{nz}}}{n} + 10}{\gamma\left(m + 4\frac{N_{\text{nz}}}{n} + 6\right)} \\ &\simeq \frac{2\rho + 6}{\gamma(\rho + 4)},\end{aligned} \tag{3.13}$$

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

$$1.5 < \frac{2\rho + 6}{\rho + 4} < 2, \tag{3.14}$$

we can draw the following assumptions:

- If $\gamma = 1.0$, indicating no convergence behavior degradation. Due to the ratio of bytes of FP64 and FP32, the maximum speedup is up to 2.0.
- If $\gamma < 1.5$, MP-GMRES(m) has the potential to accelerate.
- If $\gamma > 2.0$, which means that MP-GMRES(m) requires at least twice as many m-iteration GMRES as FP64-GMRES(m). And MP-GMRES(m) is likely to require more execution time than FP64-GMRES(m) using the same m.
- As ρ becomes larger, the expected speedup increases.

3.5 Numerical Results

In this section, comprehensive numerical experiments are conducted to illustrate the numerical behavior of MP-GMRES(m) from a practical perspective. Using various matrices in the SuiteSparse Matrix Collection [16], detailed analysis and comparison with the FP64-GMRES(m) are made from three aspects: the maximum attainable accuracy, the number of iterations for attaining convergence criterion, and the execution time. Additionally, some interest cases are discussed in this section as well.

3.5.1 Problem Settings

Consider the linear system

$$Ax = b, \tag{3.15}$$

where $A \in \mathbb{R}^{n\times n}$ is the coefficient matrix and is large, sparse, and non-symmetric, which often occurs in practice. $x \in \mathbb{R}^n$ is the solution vector and $b \in \mathbb{R}^n$ is the right-hand side vector.

With mixed precision computing techniques, most computations in MP-GMRES(m) are performed in FP32, while the input for the original problem and the output for the approximate solution are maintained in FP64.

For both FP64-GMRES(m) and MP-GMRES(m) methods, we use relative residual 2-norm, defined as

$$\frac{\|r\|_2}{\|b\|_2} \quad or \quad \frac{\|b - Ax\|_2}{\|b\|_2}, \tag{3.16}$$

to check whether the convergence criterion is satisfied, where both data and computations are conducted in FP64.

3.5.2 Experiment Settings

The FP64-GMRES(m) and MP-GMRES(m) methods are experimented on a single computational node of the supercomputer system Grand Chariot operated at Hokkaido University. The specific configuration of the platform is shown in Table 3.3. The operating system is Red Hat Enterprise Linux Server release 7.6. The programming language is C language, and the compilation environment is `icc` (ver. 19.1.3.304). Besides, `OpenMP` is used for parallelization.

The experiments include three parts of evaluation: the maximum attainable accuracy, the total number of inner iterations, and the execution time. Among them, maximum attainable accuracy and the number of iterations are evaluated using serial code. On the other hand, the evaluation of execution time is performed using OpenMP parallelism, where SpMV and MGS kernels in the Arnoldi process are paralleled using OpenMP.

Selected from the SuiteSparse Matrix Collection, the basic information of the matrices is shown in Table 3.4. In order to investigate the numerical behavior of the MP-GMRES(m) method on various problems, 26 matrices whose 2-norm condition number $\kappa_2(A)(= \|A\|_2\|A^{-1}\|_2)$ in the range from $O(10^1)$ to $O(10^8)$ are selected, and are stored in the CSR format in the experiments. Out of these 26 matrices, the value of N_{nz}/n of 23 matrices is smaller than 50, which is the minimum value of restart frequency in this research. The value of the other three matrices is smaller than 150. These data basically satisfy the condition (3.12) in our experiments.

The other settings in the experiment are listed as follows.

- Initial guess: $x_0 = [0, 0, \ldots, 0]^\top$
- Right-hand side vector: $b = [1, 1, \ldots, 1]^\top$

Table 3.3: Information of the computational platform used in the numerical experiments.

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

- Convergence criterion: $\frac{\|b - Ax\|_2}{\|b\|_2} \leq 10^{-10}$
- Iteration limit: (total number of inner iterations) $\leq n$
- Candidates of m: 50, 100, 200, 300, 400, 500

And the relative residual 2-norm is computed with A, b, and x in FP64, which is the same as that in FP64-GMRES(m). And the total number of iterations is the sum of iterations in m-iteration GMRES in each restart.

3.5.3 Analysis on the Maximum Attainable Accuracy

By introducing mixed precision computing, whether MP-GMRES(m) is still able to solve the problems that FP64-GMRES(m) can solve is one of our concerns. Table 3.5 provides the maximum attainable relative residual 2-norm in FP64-GMRES(m) and MP-GMRES(m). In the table, we use $\log_{10}(\|b - Ax\|_2/\|b\|_2)$ to show the results, and "-11" means that the method attains the convergence criterion within iteration limits.

From Table 3.5, FP64-GMRES(m) and MP-GMRES(m) both converge or do not converge in 23 out of 26 matrices. In three cases (`airfoil_2d`, `chipcool1` and `TSOPF_RS_b39_c7`), only the FP64-GMRES(m) method can converge. However, when both two methods fail to converge, there is almost no difference in the final attainable accuracy. It is worth noting that in some cases with large condition numbers, the initial solution is not greatly improved after a series of iterations. This suggests the potential necessity for additional optimization techniques, such as preconditioning, which is one of our directions in further research.

3.5.4 Analysis on the Total Number of Iterations

For each matrix, experiments are conducted with each m. The results of FP64-GMRES(m) and MP-GMRES(m) are shown in Tables 3.6 and 3.7, respectively. In the tables, "—" represents that the method did not satisfy the convergence criterion within the iteration limit.

First and foremost, both of the FP64-GMRES(m) method and MP-GMRES(m) method can converge, except for `airfoil_2d`, `chipcool1` and `TSOPF_RS_b39_c7`.

Table 3.4: Basic information of the test matrices selected from the SuiteSparse Matrix Collection.

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

Table 3.5: Attainable accuracy of FP64-GMRES(m) and MP-GMRES(m) within the iteration limit: $\lfloor \log_{10}(\|b - Ax\|_2/\|b\|_2) \rfloor$ when the solver attains 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	-1	-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

This result indicates that if practice problems can be solved by FP64-GMRES(m), MP-GMRES(m) is also expected to solve the problems.

Next, for the same restart frequency m, MP-GMRES(m) method requires almost the same (especially when m is small) or even more number of iterations as FP64-GMRES(m) does in most cases. This is consistent with the theoretical analysis in Section 3.4.1.

Moreover, the increase of m also impacts the trend of the total number of iterations. When m is small, the total numbers of iterations of the FP64-GMRES(m) and MP-GMRES(m) methods are basically the same. However, as m increases, there is a difference between the two methods. For FP64-GMRES(m), the number of iterations decreases as m increases, while for MP-GMRES(m), an increase sometimes occurs(e.g., `cage10`, `poisson3Da`). We speculate that this trend is related to the decrease in the relative residual 2-norm after each restart, which is possibly influenced by two factors: the difference in computational capability between FP64 and FP32, and the matrix itself.

Take `memplus` for an example, Figure 3.1 shows the change of the total number of iterations in FP64-GMRES(m) and MP-GMRES(m). This figure shows that the number of iterations decreases in FP64-GMRES(m), whereas, in MP-GMRES(m), it first decreases and then increases. This trend is consistent with the previous discussion. Additionally, when $m = 50$ and 100, MP-GMRES(m) requires less number of iterations which need further study.

Furthermore, in addition to the general trends mentioned above, there are also some special cases. For `cage10` and `Zhao1`, FP64-GMRES(m) converges within one restart. However, no such cases in MP-GMRES(m) are found in our experiments. Due to the fact that the computation is performed in FP32 in one restart, this result is acceptable.

3.5.5 Analysis on the Execution Time

Table 3.8 shows the performance of FP64-GMRES(m) and MP-GMRES(m) in terms of the execution time. OpenMP parallelism is used in these experiments, and the number of threads is 40. To better compare the execution time, only cases where both FP64-GMRES(m) and MP-GMRES(m) converge are presented in the table. And the underlined data is the shortest execution time of each method among different m of one matrix.

Compared among different values of m in one algorithm, both FP64-GMRES(m) and MP-GMRES(m) show short execution time when m is small, and even provide the shortest execution time in some cases.

Compared between FP64-GMRES(m) and MP-GMRES(m), the MP-GMRES(m) method is faster than FP64-GMRES(m) method in most cases. When m is small, the number of iterations of FP64-GMRES(m) and MP-GMRES(m) are almost the same, as analyzed in Section 3.5.4, and MP-GMRES(m) shows an advantage in execution time.

Additionally, the superiority of MP-GMRES(m) is reflected by the fact that although more iterations are needed in MP-GMRES(m) in some cases, its execution time is still shorter. Take `wang4` as an example; the shortest execution

Table 3.6: Total number of inner iterations in FP64-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	100	200	300	400	500
$O(10^1)$	cage10	26	26	26	26	26	26
$O(10^2)$	appu	114	110	108	108	108	108
	Zhao1	43	43	43	43	43	43
	FEM_3D_thermal1	318	270	260	250	250	250
	ns3Da	2,310	1,983	1,993	1,791	1,945	1,522
$O(10^3)$	poisson3Da	306	260	184	184	184	184
	wang3	937	670	486	319	313	313
	epb1	1,994	1,781	1,516	1,290	1,153	1,095
$O(10^4)$	coupled	—	—	1,846	1,450	1,025	872
	af23560	—	—	—	4,799	4,305	4,442
	Zhao2	—	3,878	3,488	2,868	2,880	2,929
$O(10^5)$	memplus	5,900	3,030	1,942	1,532	1,459	1,352
	wang4	—	—	1,741	1,101	738	688
	viscoplastic2	—	2,061	1,530	1,409	1,375	1,299
$O(10^6)$	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	—	—	—	—	—	—

Table 3.7: Total number of inner iterations in 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	100	200	300	400	500
$O(10^1)$	cage10	59	110	211	311	411	511
$O(10^2)$	appu	114	151	251	351	451	551
	Zhao1	73	123	223	323	423	523
	FEM_3D_thermal1	318	270	300	399	511	611
	ns3Da	2,316	1,984	1,989	1,790	1,945	1,522
$O(10^3)$	poisson3Da	320	270	446	651	833	1,018
	wang3	896	663	587	719	957	1,150
	epb1	1,881	1,786	1,517	1,302	1,153	1,272
$O(10^4)$	coupled	—	9,873	2,094	1,441	1,091	1,324
	af23560	—	—	—	4,777	4,283	4,351
	Zhao2	—	3,611	3,494	2,846	2,880	2,932
$O(10^5)$	memplus	4,521	2,943	2,044	2,404	3,265	3,108
	wang4	—	—	1,896	1,423	1,477	1,792
	viscoplastic2	—	1,969	1,658	1,700	2,046	2,274
$O(10^6)$	airfoil_2d	—	—	—	—	—	—
	inlet	—	—	—	—	—	—
	jan99jac040sc	—	—	—	—	—	—
	chipcool1	—	—	—	—	—	—
$O(10^7)$	TSOPF_RS_b39_c7	—	—	—	—	—	—
	sme3Da	—	—	—	—	—	—
	garon2	—	—	—	—	—	—
	shermanACb	—	—	—	—	—	—
$O(10^8)$	powersim	—	—	—	—	—	—
	circuit_3	—	—	—	—	—	—
	e40r0100	—	—	—	—	—	—
	rajat15	—	—	—	—	—	—

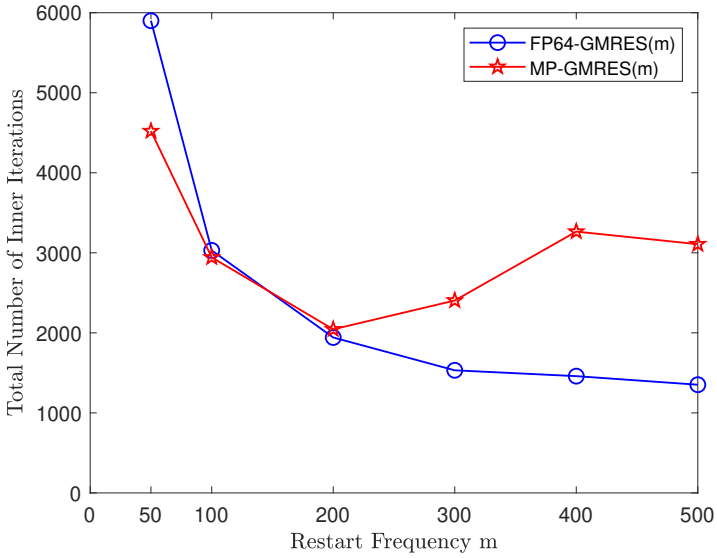


Figure 3.1: The change of the total number of inner iterations in FP64-GMRES(m) and MP-GMRES(m) for `memplus`.

time of FP64-GMRES(m) and MP-GMRES(m) is 3.34×10^0 and 1.69×10^0 while the corresponding numbers of iterations are 738 and 1896, respectively. About 2.6 times the iteration is required in MP-GMRES(m), MP-GMRES(m) is still faster.

The speedup ratio is used to better evaluate the impact of mixed precision computing and compare FP64-GMRES(m) with MP-GMRES(m). Here, the comparison of the speedup ratio of MP-GMRES(m) over FP64-GMRES(m) between the actual and estimated results is listed in Table 3.9. Only the cases where FP64-GMRES(m) and MP-GMRES(m) obtain the shortest execution time with the same m are included. It is validated against the previously mentioned theoretical model and indicates the consistency of the experimental results with the theoretical analysis.

In this research, the actual speedup is computed as

$$(\text{Speedup}) = \frac{Execution\ time\ of\ FP64-GMRES(m)}{Execution\ time\ of\ MP-GMRES(m)}. \tag{3.17}$$

And the estimated speedup is calculated using (3.13), where γ uses the actual number of iterations in Tables 3.6 and 3.7.

As seen from Table 3.9, MP-GMRES(m) is able to accelerate in most cases. Meanwhile, the estimated speedup is acceptable as it does not differ much from the actual speedup in most cases. These results indicate the effectiveness of MP-GMRES(m).

Table 3.8: 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 3.9: Comparison of the speedup ratio of MP-GMRES(m) over FP64-GMRES(m) between the actual and estimated results

$\kappa_2(A)$	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

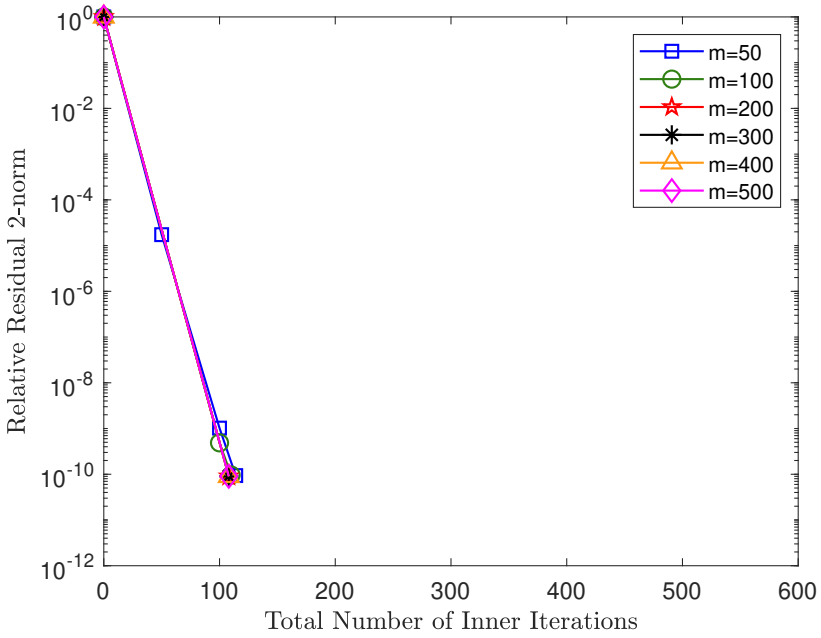
3.5.6 Case Study

This subsection presents some interesting cases and provides the history of relative residual 2-norm in some figures. A mark represents a restart in these figures, and the corresponding relative residual 2-norm is computed in FP64. The following analysis will be provided based on whether the methods converge or not.

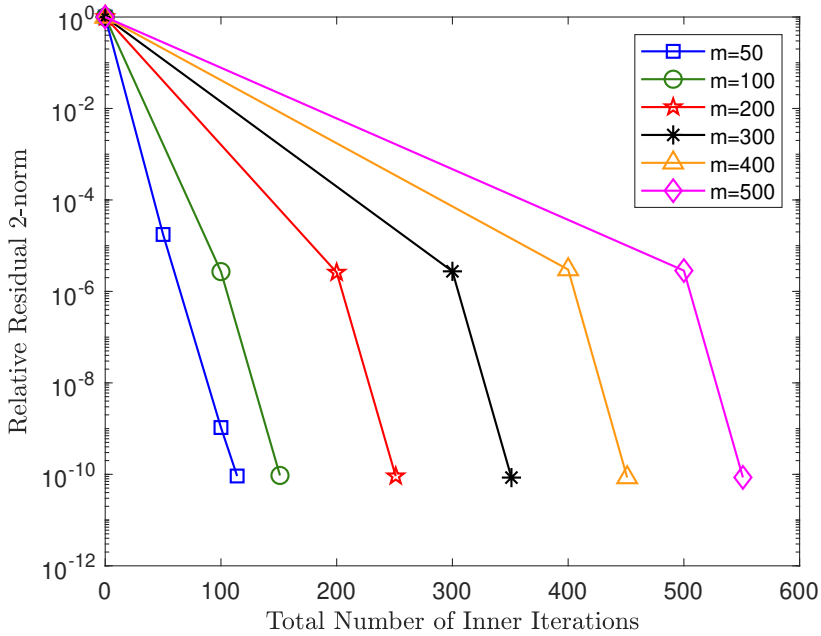
I. Cases that both FP64-GMRES(m) and MP-GMRES(m) attain the convergence criterion.

- Cases for `appu`, `memplus` and `ns3Da`. Figures 3.2, 3.3 and 3.4 show the history of relative residual 2-norm for `appu`, `memplus`, `ns3Da`, respectively. It is observed that the numbers of iterations of FP64-GMRES(m) decrease as m increases. However, the number of iterations of MP-GMRES(m) increases in some cases. For example, it keeps on increasing when m increases for `appu`. For `memplus`, the number of iterations first decreases and then increases, with the minimum value occurring at $m = 200$ (Figure 3.3b). The number of iterations for `ns3Da` tends to decrease with increasing m, but it also shows a slight increase at $m = 400$.

- Cases for `appu` and `Zhao1`. FP64-GMRES(m) can converge in one restart in the cases for `appu`(Figure 3.2a) and `Zhao1`(Figure 3.5a)., e.g., $m \geq 200$ in `appu`, $m \geq 50$ in `Zhao1`. However, seen from Figures 3.2b and 3.5b, the relative residual 2-norm is reduced limitedly within the first restart(up to around 10^{-5}) in MP-GMRES(m).

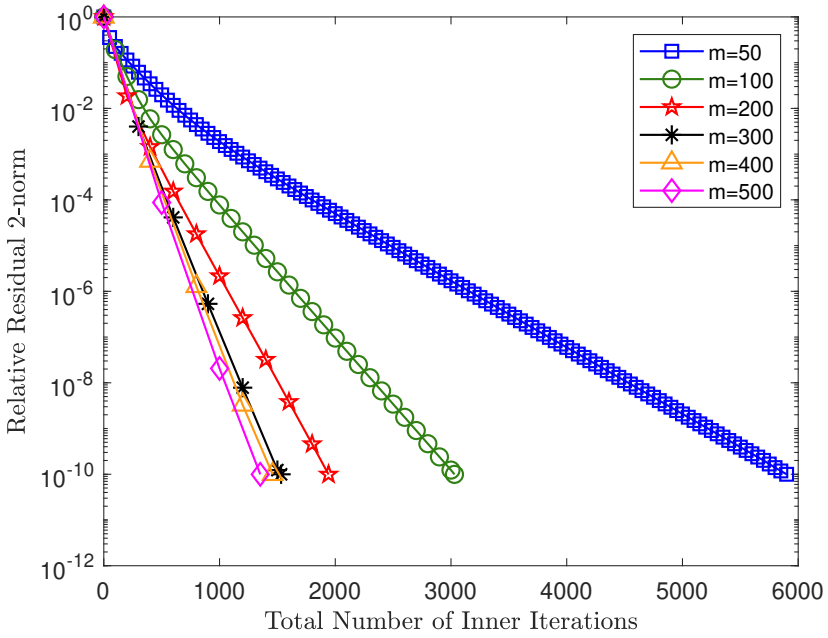


(a) FP64-GMRES(m)

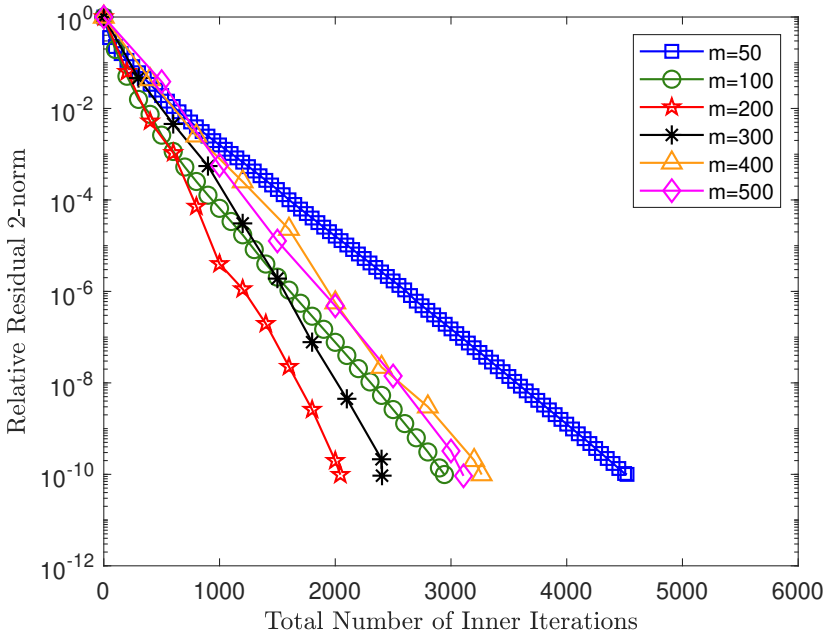


(b) MP-GMRES(m)

Figure 3.2: Relative residual 2-norm history for `appu`.

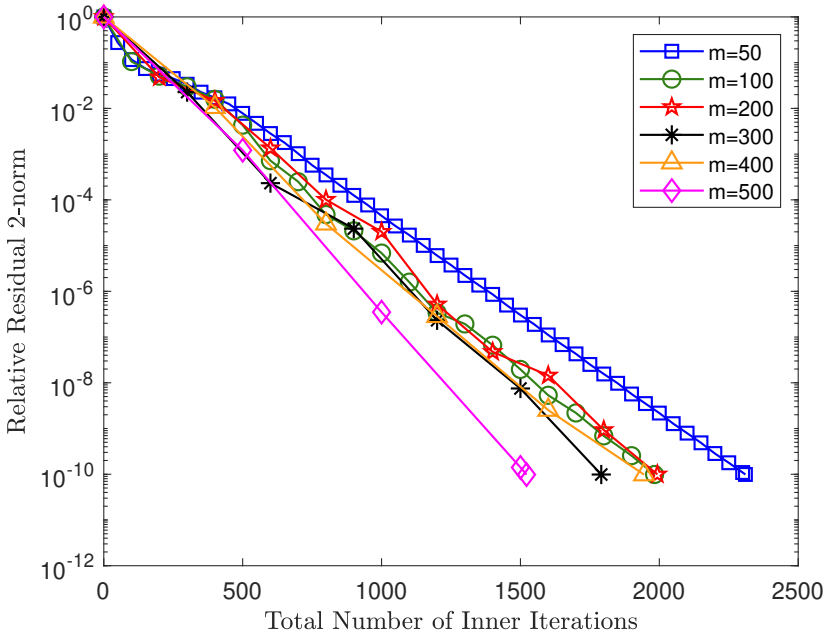


(a) FP64-GMRES(m)

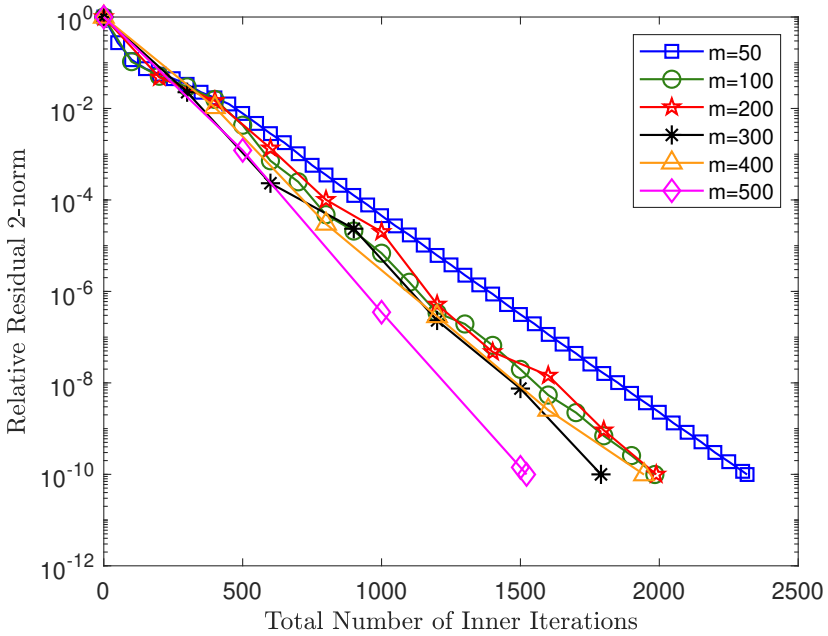


(b) MP-GMRES(m)

Figure 3.3: Relative residual 2-norm history for `memplus`.

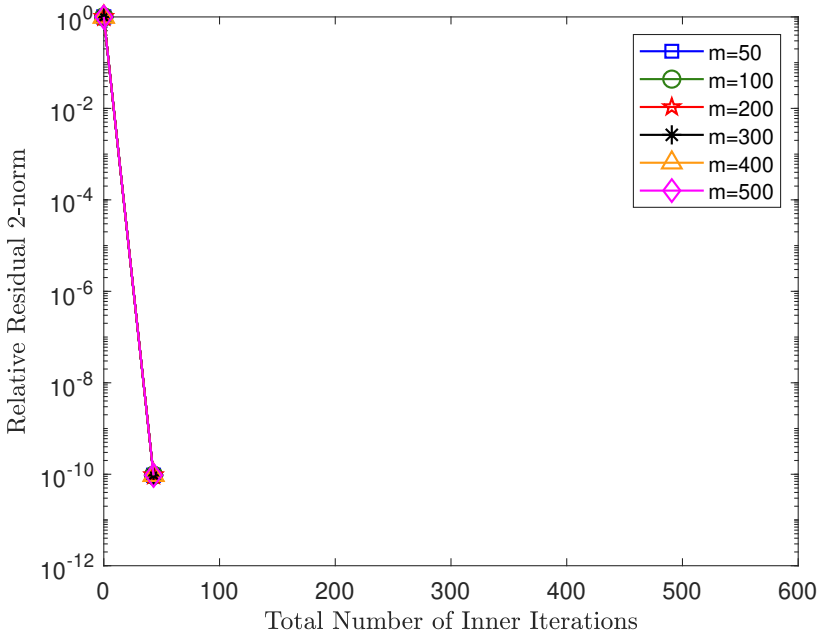


(a) FP64-GMRES(m)

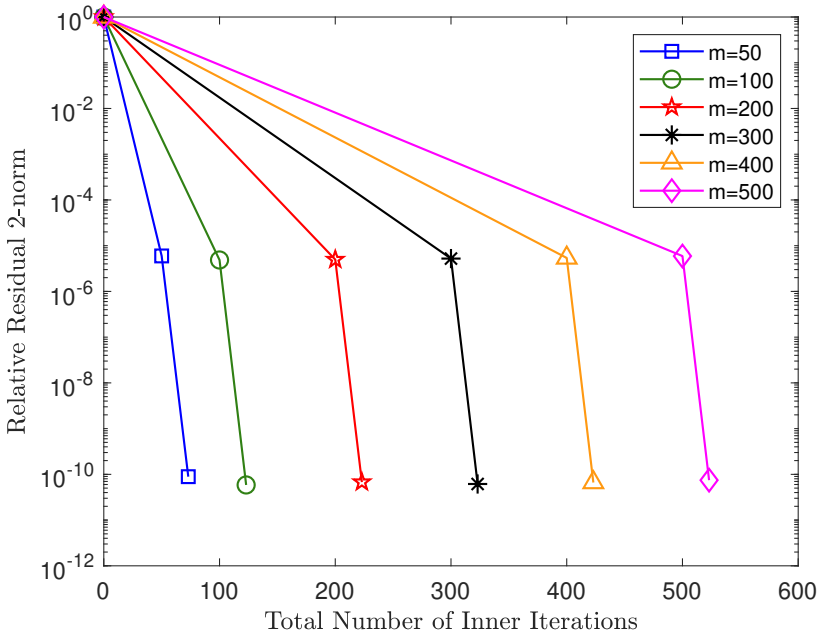


(b) MP-GMRES(m)

Figure 3.4: Relative residual 2-norm history for `ns3Da`.



(a) FP64-GMRES(m)



(b) MP-GMRES(m)

Figure 3.5: Relative residual 2-norm history for `Zhao1`.

II. Cases that FP64-GMRES(m) can attain while MP-GMRES(m) cannot attain the convergence criterion.

- Cases for `airfoil_2d` and `TSOPF_RS_b39_c7`. From Figures 3.6 and 3.7, FP64-GMRES(m) can converge while the MP-GMRES(m) cannot. However, the maximum attainable accuracy shows a different trend. From Figure 3.6b, although MP-GMRES(m) cannot converge in `airfoil_2d`, the attainable accuracy is very close to the convergence criterion. Thus, it is expected to converge if more iterations can be performed.

  On the other hand, for `TSOPF_RS_b39_c7`, the relative residual 2-norm does not reduce in MP-GMRES(m) (Figure 3.7b). These two cases show two quite different convergence behaviors.

III. Cases that neither FP64-GMRES(m) nor MP-GMRES(m) cannot attain the convergence criterion.

- Case for `garon2`. From Figure 3.8, neither FP64-GMRES(m) nor MP-GMRES(m) cannot converge. And their histories of relative residual 2-norm follow a similar trend.

3.6 Conclusion

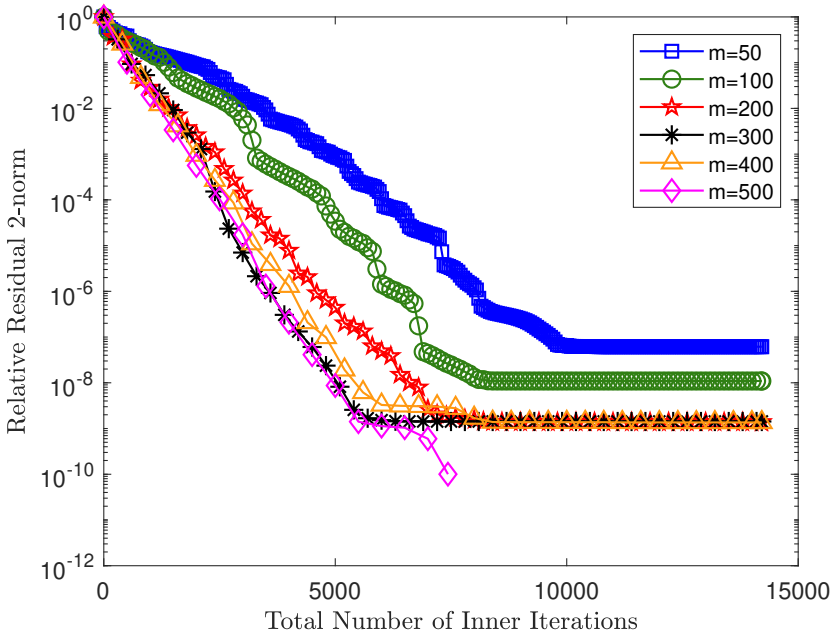
Using 26 matrices whose condition number range from $O(10^1)$ to $O(10^8)$ as experimental subjects, experiments are conducted in both FP64-GMRES(m) and MP-GMRES(m) methods. By observing and analyzing three characteristics (the maximum attainable accuracy, the numerical behavior, and execution time) with various matrices and different settings of m, the numerical behavior and effectiveness of the MP-GMRES(m) are clarified. Specifically, the impact of m and the history of relative residual 2-norm are also investigated.

From the experimental results and analysis, one important conclusion can be drawn that if a problem can be solved by FP64-GMRES(m), MP-GMRES(m) can also be effective except for a few cases. This is crucial for us to consider the replacement of FP64-GMRES(m) with MP-GMRES(m) in practical applications. Through this study, for problems that have been successfully solved by FP64-GMRES(m), adopting the MP-GMRES(m) method is a highly worthwhile consideration to enhance problem-solving efficiency.

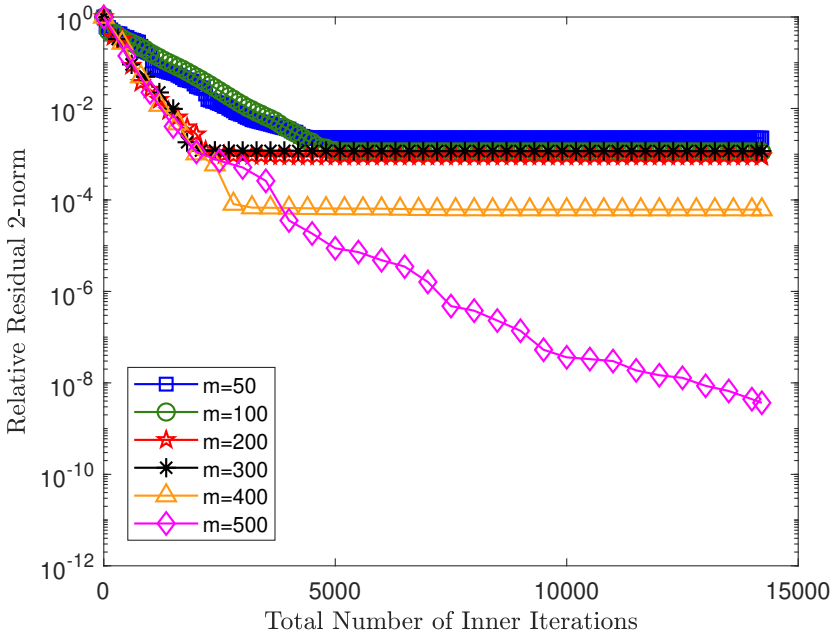
Besides, our study also obtains other conclusions. As mentioned before, MP-GMRES(m) is not a new method and has been thoroughly studied by many scholars [64, 3, 4, 44, 45]. Some of our results are consistent with some previous studies, while others bring some new findings.

The consistent results are as follows.

- Convergence: it is observed that if a problem can be solved by the FP64-GMRES(m), the MP-GMRES(m) method can generally solve it as well.

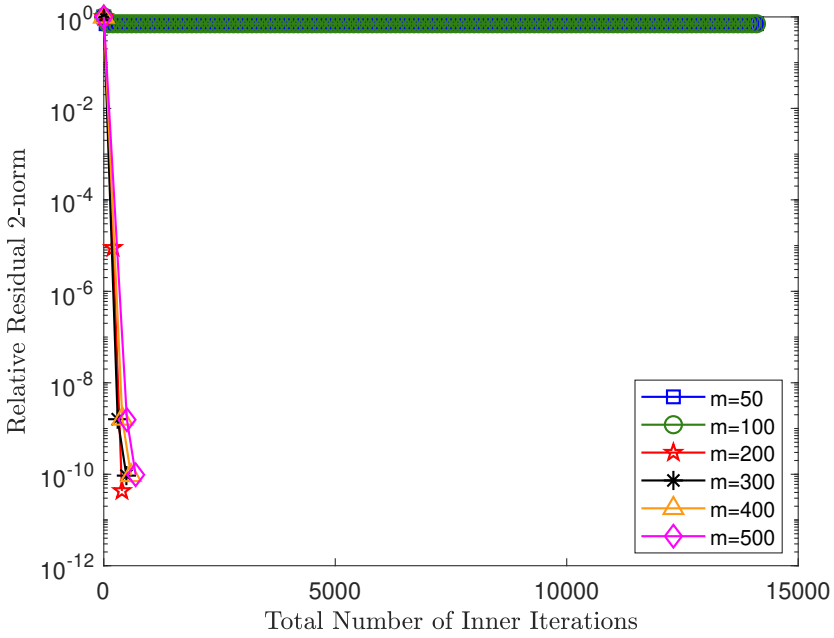


(a) FP64-GMRES(m)

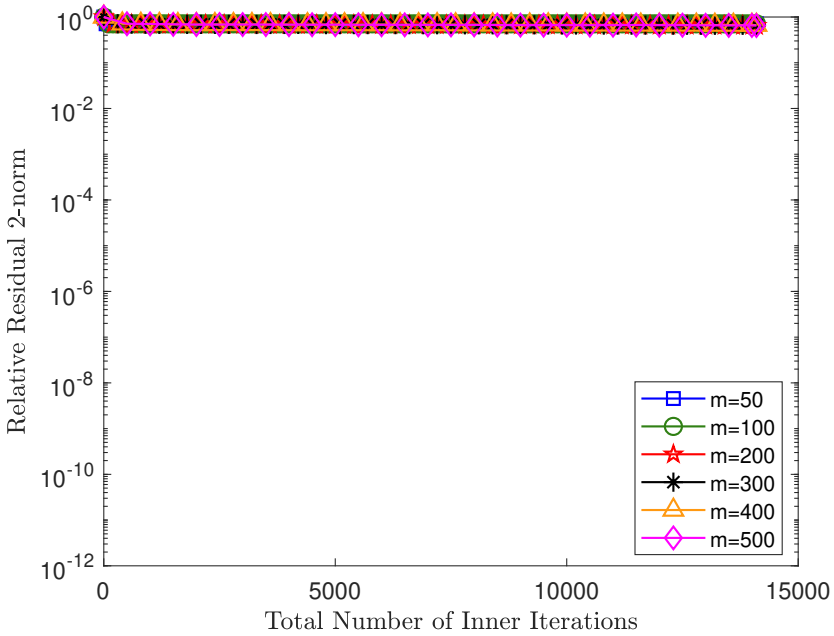


(b) MP-GMRES(m)

Figure 3.6: Relative residual 2-norm history for `airfoil_2d`.

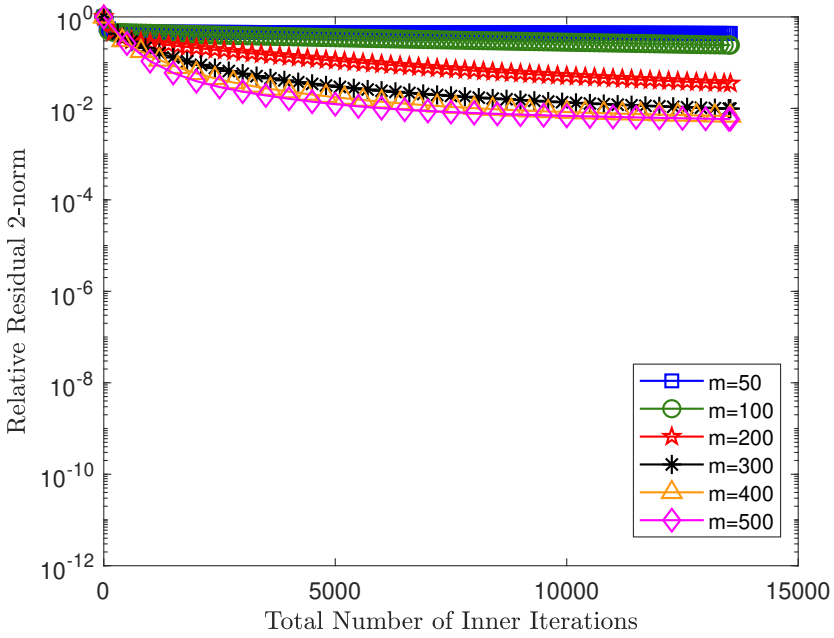


(a) FP64-GMRES(m)

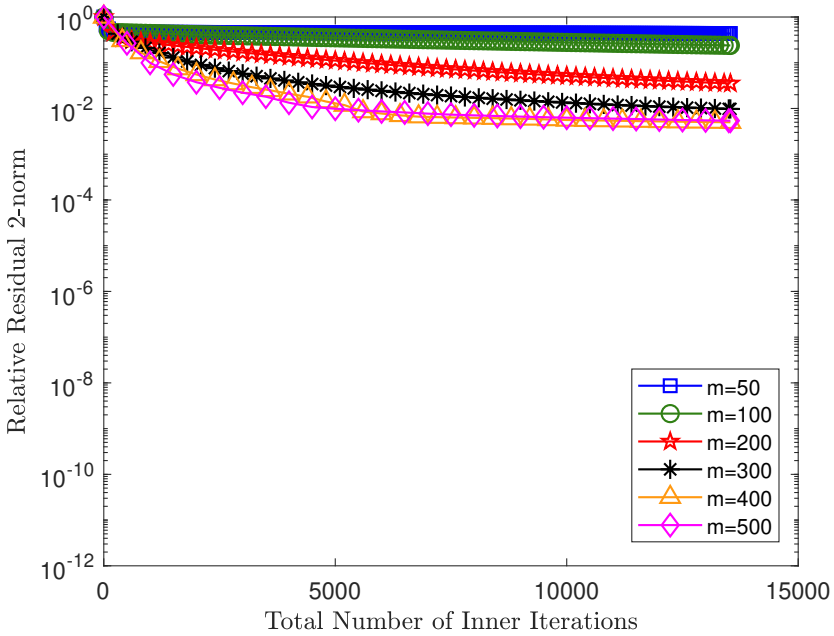


(b) MP-GMRES(m)

Figure 3.7: Relative residual 2-norm history for `TSOPF_RS_b39_c7`.



(a) FP64-GMRES(m)



(b) MP-GMRES(m)

Figure 3.8: Relative residual 2-norm history for `garon2`.

- The number of iterations: when m is small, the number of iterations of MP-GMRES(m) is almost the same as that of FP64-GMRES(m). For example, the results with $m = 10$ in [64, 3], as well as the results in our study (e.g., $m = 50$, $m = 100$ for `FEM_3D_thermal1`, `epb1`).

  Besides, FP64-GMRES(m) can attain the convergence criterion within one restart, while MP-GMRES(m) requires more iterations. More details can be seen in the case of `cage10`, `Zhao1` in our study and paper [44].

- Execution time: MP-GMRES(m) is faster than FP64-GMRES(m), which is consistent with the research in the papers [64, 3, 4, 44]

Additionally, our research also obtained some new findings:

- Almost no difference in maximum attainable accuracy.

  MP-GMRES(m) method is able to maintain the same attainable accuracy as FP64-GMRES(m) in most cases. By analyzing the history of relative residual 2-norm, the accuracy is compared. In cases where FP64-GMRES(m) and MP-GMRES(m) both converge or do not converge, their accuracy is basically the same. However, for some matrices, the MP-GMRES(m) may fail to converge when FP64-GMRES(m) converges, and the convergence history also shows differences.

- Theoretical and practical analysis on the number of iterations and execution time.

  The characteristics of MP-GMRES(m) are analyzed from theoretical and practical aspects. And the theoretical analysis and experimental results are generally consistent. The MP-GMRES(m) requires the same or more number of iterations and it is faster than FP64-GMRES(m) in most cases. In particular, in some cases, MP-GMRES(m) is still faster than FP64-GMRES(m) even if it requires more iterations.

- The impact of m: different trends of numerical behavior as m increases.

  The changing trend of numerical behavior is analyzed with different settings of m. In terms of the number of iterations, they are basically the same in FP64-GMRES(m) and MP-GMRES(m) when m is small. As m increases, the trends are different; the number of iterations decreases in FP64-GMRES(m), while that in MP-GMRES(m) sometimes increases.

  As for execution time, except for some problems that can only be solved by a larger m, most of the problems basically achieve the fastest execution time with a small m.

- Typical cases are studied via relative residual 2-norm. These illustrate the convergence process of the methods, which helps us better understand and optimize the method.

Our study demonstrates the potential of MP-GMRES(m) and provides a detailed investigation of MP-GMRES(m), as well as the comparison with FP64-GMRES(m). In our study, an in-depth understanding of the restart parameter m, performance, and applicability are also provided, which gives new insights for further study and optimization.

Chapter 4

Numerical Investigation of Mixed Precision Iterative Refinement using the BiCGSTAB Method

In this chapter, the BiCGSTAB method and its employment in mixed precision iterative refinement are studied. First, a brief introduction is given. Then, the detail of BiCGSTAB and how it is employed in mixed precision iterative refinement is introduced. Next, comparisons among different algorithms, which also include MP-GMRES(m), are made. Finally, the results are summarized and conclusions are given.

4.1 Introduction

The BiConjugate Gradient STABilized (BiCGSTAB) method is another famous Krylov subspace method used to solve large, sparse, and non-symmetric linear systems. Although the BiCGSTAB method seems more sensitive and unstable to a problem than the GMRES(m) method, it is known to be as effective and fast as the GMRES(m) method for certain problems. Inspired by MP-GMRES(m) method, whether mixed precision computing and the BiCGSTAB method can be effectively combined and then obtain improvement is a challenging and interesting topic. Currently, few studies on this topic have been reported [15], only an attempt in an application. And we conduct research to explore its possibility and effectiveness.

In this research, a new mixed precision algorithm is developed based on the mixed precision iterative refinement (MP-IR), namely mixed precision iterative refinement using BiCGSTAB (MP-IR using BiCGSTAB). Similar to the research on MP-GMRES(m) in Chapter 3, the new mixed precision algorithm is investi-

gated, and its numerical characteristics are evaluated in terms of the attainable accuracy, the number of iterations, and the execution time. Additionally, the comparison with MP-GMRES(m) is also given. Some interest cases are also described.

There are two differences in the MP-IR using BiCGSTAB compared with the traditional BiCGSTAB method: the iterative refinement and low precision computing. Therefore, three sets of experiments are respectively conducted on BiCGSTAB, FP64-IR using BiCGSTAB, and MP-IR using BiCGSTAB to better study the impacts of these two factors.

The study in Chapter 4 helps to deepen the researchers' understanding of the employment of the BiCGSTAB method in mixed precision iterative refinement and also provides a reference for further optimization and application of the algorithm. Besides, the comparison between MP-IR using BiCGSTAB and MP-GMRES(m) helps select an appropriate method based on the target problem in practice applications.

4.2 Review of the BiCGSTAB method and its Employment in Mixed Precision Iterative Refinement

In this section, the introduction of the BiCGSTAB method is first given. Then, how BiCGSTAB is employed in mixed precision iterative refinement is explained in detail.

4.2.1 The BiCGSTAB Method

The BiConjugate Gradient STABilized(BiCGSTAB) method [66] is another iterative method used to solve large, sparse, and non-symmetric linear systems. The method was proposed by van der Vorst [65] in 1992 and is a variant of the biconjugate gradient method (BiCG) [20, 8, 58, 66]. Both the BiCGSTAB and BiCG methods belong to the Krylov subspace methods. Next, a brief introduction to the BiCG method is given, followed by an explanation of how BiCGSTAB is derived from the BiCG method.

The brief steps of the BiCG method are as follows:

Step 1: Construct two Krylov subspaces.

Let x_0 be an initial guess and construct two Krylov subspaces $\mathscr{K}_k(A, r_0)$ and $\mathscr{K}_k(A^\top, r_0^*)$, where $\mathscr{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$ and $\mathscr{K}_k(A^\top, r_0^*) = \text{span}\{r_0^*, A^\top r_0^*, (A^\top)^2 r_0^*, \ldots, (A^\top)^{k-1} r_0^*\}$. In the subspaces, $r_0 = b - Ax_0$ and r_0^* is an arbitrary vector satisfying $r_0^\top r_0^* \neq 0$. The BiCG method then can be described as finding an approximate solution $x_k \in x_0 + \mathscr{K}_k(A, r_0)$ that satisfies the condition $r_k \perp \mathscr{K}_k(A^\top, r_0^*)$, where $r_k = b - Ax_k$.

Step 2: Obtain matrices V_k and W_k, and a tridiagonal matrix $\tilde{T}_k$ by the BiLanczos process.

The BiLanczos process is the process of simultaneously finding the basis of subspaces such that the two sets of bases are orthogonal to each other. Through BiLanczos process, two matrices $V_k = \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix} \in \mathbb{R}^{n\times k}$ and $W_k = \begin{bmatrix} w_1 & w_2 & \cdots & w_k \end{bmatrix} \in \mathbb{R}^{n\times k}$ are obtained, where column vectors respectively form an orthogonal basis of $\mathscr{K}_k(A, r_0)$ and $\mathscr{K}_k(A^\top, r_0^*)$. And a $(k+1)$-by-k tridiagonal matrix

$$\tilde{T}_k = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \gamma_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \gamma_{k-1} & \alpha_k \\ & & & \gamma_k \end{pmatrix} \in \mathbb{R}^{(k+1)\times k} \tag{4.1}$$

is also obtained. And V_k, W_k and T_k satisfy the relation

$$W_k^\top A V_k = T_k, \tag{4.2}$$

where $T_k \in \mathbb{R}^{k\times k}$ is the matrix consisting of the first k row of the matrix $\tilde{T}_k$.

Step 3: Solve miniaturization problem.

Through the BiLanczos process in step 2, the approximate solution can be expressed as

$$x_k = x_0 + V_k y_k, \tag{4.3}$$

and the problem reduces to finding a vector y_k that satisfy $r_k \perp \mathscr{K}_k(A^\top, r_0^*)$. Then, we have

$$\begin{aligned} 0 &= W_k^\top r_k \\ &= W_k^\top (b - A x_k) \\ &= W_k^\top (b - A(x_0 + V_k y_k)) \\ &= W_k^\top (r_0 - A V_k y_k) \\ &= W_k^\top r_0 - W_k^\top A V_k y_k. \end{aligned} \tag{4.4}$$

Using the relation (4.2), we have

$$W_k^\top r_0 - W_k^\top A V_k y_k = 0 \Leftrightarrow \beta W_k^\top v_1 - T_k y_k = 0 \tag{4.5}$$

Thus, y_k is the solution of the following linear system

$$T_k y_k = \beta W_k^\top v_1, \tag{4.6}$$

which can be solved by LU factorization.

However, the BiCG method is generally less stable than the GMRES method. The residual 2-norm does not decrease all the time during the iteration process. Thus, the convergence curve is irregular, which brings difficulties in investigating the convergence. Besides, $A^\top$ is used in the BiCG method, but in practical

applications, $A^\top$ or the product of $A^\top$ and a vector are not easy to obtain. This also brings difficulty in implementing the method. Therefore, we would like to avoid using $A^\top$ as much as possible during the computing.

As a variant of the BiCG method, the BiCGSTAB method [66], proposed by H. A. van der Vorst [65], gains improvements in stability and convergence speed. The basic idea is presenting the k-th residual vector as

$$r_k = Q_k(A) r_k^{\mathrm{BiCG}}, \tag{4.7}$$

where r_k^{BiCG} is the k-th residual vector in the BiCG method, and $Q_k(t)$ is a polynomial of degree k, which is used to correct the residual 2-norm.

In the BiCGSTAB method, $Q_k(t)$ is defined as

$$Q_k(t) = (1 - \zeta_{k-1} t) Q_{k-1}(t), \quad Q_0(t) = 1, \tag{4.8}$$

where ζ_{k-1} is obtained so as to

$$\begin{aligned} \min_{\zeta_{k-1}} \|r_k\|_2 &\Leftrightarrow \min_{\zeta_{k-1}} \|Q_k(A) r_k^{\mathrm{BiCG}}\|_2 \\ &\Leftrightarrow \min_{\zeta_{k-1}} \|(I - \zeta_{k-1} A) Q_{k-1}(A) r_k^{\mathrm{BiCG}}\|_2. \end{aligned} \tag{4.9}$$

The outline of the BiCGSTAB method is shown in Algorithm 5. In Algorithm 5, $(\cdot, \cdot)$ is the inner product of two vectors, e.g., $(x, y) = x^\top y$.

Algorithm 5 The BiCGSTAB method

Input: x_0: initial guess, A; coefficient matrix, b: right-hand side vector, ϵ: convergence criterion, m: iteration limit

1: $r_0 = b - A x_0$
2: Set an arbitrary vector satisfying $(r_0^*, r_0) \neq 0$ to r_0^*. ▷ e.g., $r_0^* = r_0$
3: $\beta_{-1} = 0$
4: **for** $k = 0, 1, \ldots, m$ **do**
5: **if** $\|r_k\|_2 / \|b\|_2 \leq \epsilon$ **then return** x_k
6: $p_k = r_k + \beta_{k-1}(p_{k-1} - \zeta_{k-1} A p_{k-1})$
7: $\alpha_k = (r_0^*, r_k)/(r_0^*, A p_k)$
8: $t_k = r_k - \alpha_k A p_k$
9: $\zeta_k = (A t_k, t_k)/(A t_k, A t_k)$
10: $x_{k+1} = x_k + \alpha_k p_k + \zeta_k t_k$
11: $r_{k+1} = t_k - \zeta_k A t_k$
12: $\beta_k = (\alpha_k / \zeta_k) \cdot ((r_0^*, r_{k+1})/(r_0^*, r_k))$
13: **end for**

Output: x_k: (approximate) solution vector

Compared to the BiCG method, the BiCGSTAB method shows faster and smoother convergence behavior. And it does not require $A^\top$, which is required in the BiCG method. This is also an advantage of BiCGSTAB and makes it easy to implement in a wide range of applications. In computation cost, the BiCGSTAB method contains two SPMVs ($A p_k$ and $A t_k$), as much as the BiCG method.

4.2.2 Employment of BiCGSTAB in Mixed Precision Iterative Refinement

Inspired by the MP-GMRES(m) method, the BiCGSTAB method is introduced into mixed precision iterative refinement, and a new mixed precision variant of iterative refinement using BiCGSTAB algorithm is developed, which we call MP-IR using BiCGSTAB for short. The structure of mixed precision iterative refinement is explained in section 3.2, and we choose the BiCGSTAB method to solve the error equation $Ae = r$ in Step 2 of IR or MP-IR. And the MP-IR using BiCGSTAB algorithm is obtained by applying the BiCGSTAB with low precision computing to Step 2, that is Line 6 in Algorithm 1. The MP-IR using BiCGSTAB algorithm consists of the following steps:

- **Step 1:** Compute the residual $r = b - A\tilde{x}$ in *high precision*.
- **Step 2:** Solve the error equation $Ae = r$ by *low precision* BiCGSTAB and obtain an approximate solution $\tilde{e}$ in *low precision*.
- **Step 3:** Update the solution $\tilde{x} = \tilde{x} + \tilde{e}$ in *high precision*.

It is found that MP-IR using BiCGSTAB has a nested loop structure, the same as MP-GMRES(m). We respectively use *inner* and *outer* to represent the loop in BiCGSTAB and the loop of MP-IR.

Compared with the traditional BiCGSTAB method, MP-IR using BiCGSTAB algorithm has two differences. One is the restart led by iterative refinement, and the other is low precision computing. In order to distinguish the influence of these two factors, it is necessary to conduct separate experiments for each factor. Therefore, experiments are conducted on BiCGSTAB, FP64-IR using BiCGSTAB, and MP-IR using BiCGSTAB. By comparing FP64-IR using BiCGSTAB with BiCGSTAB method, the impact of restart is evaluated, and the comparison between MP-IR using BiCGSTAB and FP64-IR using BiCGSTAB indicates the impact of low precision computing.

Another important point is the restart strategy, which also impacts the convergence. There are various restart strategies, some of which have been discussed in GMRES [44]. In the study related to BiCGSTAB, this means how to stop (inner) BiCGSTAB in one restart cycle of FP64-IR or MP-IR (Step 2). We consider two restart strategies. One is the number of iterations m, and the other is the accuracy ϵ. For each of the above algorithms, multiple sets of experiments are conducted using these two restart strategies to learn more about the performance of MP-IR using BiCGSTAB algorithm.

4.3 Theoretical Analysis of Mixed Precision Iterative Refinement using BiCGSTAB Method

In this section, the MP-IR using BiCGSTAB algorithm is analyzed from a theoretical aspect. First, the impacts that restart and low precision computing

may bring to numerical behavior are evaluated. Then, the speedup ratio of MP-IR using BiCGSTAB over MP-GMRES(m) is roughly estimated.

4.3.1 Analysis on the Numerical Behavior

MP-IR using BiCGSTAB algorithm differs from the traditional BiCGSTAB method in terms of restart and low precision computing. Here, the impacts of these two factors on the performance of the algorithm are theoretically analyzed.

From the aspect of the restart, if an algorithm can not solve the problem due to memory limitation, too slow convergence rate, and so on, using a restart can help the algorithm jump out of the current iteration and re-initialize its state. In this way, most of the memory used in the previous computations can be freed and reused in the next restart. The algorithm will continue to find the solution in another Krylov subspace, which may converge. Thus, there is a possibility for the algorithm to improve the convergence process with an appropriate restart. However, using a restart too often also tends to bring more number of iterations and reduce the convergence rate since the algorithm will return to the initial state several times [44]. Therefore, choosing the appropriate restart strategy and frequency is challenging and needs to be adjusted based on algorithms and practical applications.

In terms of low precision computing, all computations of BiCGSTAB are performed in FP32 in MP-IR using BiCGSTAB. Since the computational ability of FP32 is limited compared to FP64, the MP-IR using BiCGSTAB is expected to require more number of iterations than FP64-IR using BiCGSTAB under the same conditions.

4.3.2 Theoretical Estimation on the Expected Speedup of MP-IR using BiCGSTAB over MP-GMRES(m)

Execution time is also important in practical applications. In this subsection, a simple model is roughly developed based on theoretical analysis, and the acceleration of MP-IR using BiCGSTAB over MP-GMRES(m) is roughly estimated.

The model is built based on the assumption that both BiCGSTAB and GMRES are memory-bound, which means that the memory access cost primarily determines the computation time. This assumption is the same as the model in subsection 3.4.3.

In MP-IR using BiCGSTAB algorithm, BiCGSTAB is performed in FP32 and two SPMVs are included in one single iteration, where the sparse matrix is stored in the CSR format. In one SPMV, it requires $4N_{\text{nz}}$ from FP32 for values (`val` vector) and $4N_{\text{nz}}$ from INT4 for column indices (`col_ind` vector) for each non-zero elements, and $4n$ from INT4 for the information of the row partitioning (`row_ptr` vector) is also required. Besides, input and output vectors require $4n$ from FP32.

In MP-GMRES(m) method, the average memory access cost in m-iteration GMRES is roughly obtained in Table 3.2.

Table 4.1: Rough estimation of the memory access cost (in bytes) per iteration of the FP32-BiCGSTAB and m-iteration GMRES in FP32.

	FP32-BiCGSTAB $M_{\text{bicgstab}}^{(\text{FP32})}$	m-iteration GMRES in FP32 $M_{\text{m-gmres}}^{(\text{FP32})}$
SpMVs	$(8N_{\text{nz}} + 12n)$	$(8N_{\text{nz}} + 12n)$
MGS		$4 \cdot \frac{1}{2}m^2 n \cdot \frac{1}{m}$
Total	$2(8N_{\text{nz}} + 12n)$	$8N_{\text{nz}} + 12n + 2mn$

Table 4.1 presents the rough estimation of the memory access cost per iteration of the FP32-BiCGSTAB and m-iteration GMRES in FP32.

Let $\gamma = N_{\text{nz}}/n$ donate the average of the number of nonzero elements per row; the acceleration of MP-IR using BiCGSTAB over MP-GMRES(m) in one iteration is estimated as follows

$$\begin{aligned}(\text{Speedup}) &= \frac{M_{\text{bicgstab}}^{(\text{FP32})}}{M_{\text{m-gmres}}^{(\text{FP32})}} \\ &= \frac{2(8N_{\text{nz}} + 12n)}{8N_{\text{nz}} + 12n + 2mn} \\ &= \frac{8\gamma + 12}{4\gamma + m + 6}. \end{aligned} \tag{4.10}$$

From (4.10), we can obtain

$$\frac{M_{\text{bicgstab}}^{(\text{FP32})}}{M_{\text{m-gmres}}^{(\text{FP32})}} \leq 1 \Leftrightarrow 4\gamma + 6 \leq m. \tag{4.11}$$

This relation indicates that the computation cost per iteration in FP32-BiCGSTAB is expected to be smaller than m-iteration GMRES using FP32 if inequality (4.11) holds. Therefore, we can obtain the following assumptions:

- If (4.11) holds and MP-IR using BiCGSTAB requires less number of iterations than MP-GMRES(m), its execution time is expected to be shorter.
- If (4.11) does not hold and MP-IR using BiCGSTAB requires more number of iterations than MP-GMRES(m), the execution time of MP-GMRES(m) is expected to be shorter.

4.4 Numerical Results

The experiments on BiCGSTAB, FP64-IR using BiCGSTAB, and MP-IR using BiCGSTAB are conducted on a CPU platform. The obtained numerical results

are first analyzed and then compared with those of the MP-GMRES(m) method. Some interest cases are also presented. It is worth noting that the problem settings in the research on BiCGSTAB are the same as those on MP-GMRES(m) in subsection 3.5.1.

4.4.1 Experiment Settings

The BiCGSTAB method with FP64, FP64-IR using BiCGSTAB with FP64, and MP-IR using BiCGSTAB with FP64 and FP32 are experimented with multiple matrices. These three algorithms are investigated from the viewpoints of the maximum attainable accuracy, total number of iterations, and execution time. The program is executed in serial mode in the experiments on the maximum attainable accuracy and total number of iterations. For the evaluation of the execution time, `OpenMP` parallelization is used, where SPMV is parallelized.

All experiments are conducted on a single computational node of the supercomputer system Grand Chariot operated at Hokkaido University, which is the same as the previous research on MP-GMRES(m). The specific configuration of the platform is shown in Table 3.3 in section 3.5.2. The operating system is Red Hat Enterprise Linux Server release 7.6. The programming language is C language, and the compilation environment is `icc` (ver. 19.1.3.304). In addition, the option `-qopenmp` is used for parallel implementations.

The basic information of the test matrices is given in Table 3.4 in section 3.5.2. In order to better investigate the numerical characteristics and applicability of the MP-IR using BiCGSTAB, 26 matrices are selected from the SuiteSparse Matrix Collection and their condition numbers are in the range from $O(10^1)$ to $O(10^8)$. The other settings are as follows.

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

Here, the convergence criterion corresponds to ϵ in Line 4 of Algorithm 1. In the BiCGSTAB method with FP64, the total number of iterations means the iteration number for a single BiCGSTAB execution. To better compare with MP-GMRES(m) method, the test matrices and settings used in experiments are also the same as our previous experiment on MP-GMRES(m)

In FP64-IR using BiCGSTAB and MP-IR using BICGSTAB algorithms, BiCGSTAB is used to solve the error equation $Ae = r$, and we call inner BiCGSTAB or FP32-BiCGSTAB for distinction. In inner BiCGSTAB, the initial guess is set as $e_0 = [0, 0, \ldots, 0]^\top$. Two stopping conditions for the inner BiCGSTAB (or the restart strategies for FP64-/MP-IR using BiCGSTAB algorithms) are considered.

1. One is the limit of the iterations, represented by m_{in}. The candidates of m_{in} are 50, 100, 200, 300, 400, 500.

2. The other is the accuracy, represented by ϵ_{in}, which is the decrease of the relative residual 2-norm. The candidates of ϵ_{in} are 10^{-10}, 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}.

Experiments are separately carried out using these two restart strategies. In the experiment on m_{in}, $\epsilon_{\text{in}} = 10^{-10}$, and the total number of iterations is the sum of inner iterations of each inner BiCGSTAB in FP64-/MP-IR using BiCGSTAB algorithms. In the experiment on ϵ_{in}, $m_{\text{in}} = n$, the total number of iterations is the number of inner iterations of each inner BiCGSTAB in FP64-/MP-IR using BiCGSTAB algorithms.

4.4.2 Analysis on the Maximum Attainable Accuracy

The problem-solving ability of the algorithm is one of the concerns after introducing low precision computing. Figure 4.1 provides the maximum attainable accuracy of FP64-GMRES(m), MP-GMRES(m), BiCGSTAB and MP-IR using BiCGSTAB. We use $\log_{10}(\|b - Ax\|_2/\|b\|_2)$ to show the results, and the data is rounded toward negative infinity. "$\log_{10}(\|b - Ax\|_2/\|b\|_2) \leq -11$" means that the method attains the convergence criterion within the iteration limits.

From the figure, BiCGSTAB and MP-IR using BiCGSTAB can attain the same or similar convergence results in 23 out of 26 matrices. In this case, BiCGSTAB and MP-IR using BiCGSTAB can reach "-11" in 10 of them, while neither can converge in 13 of them. Besides, compared with GMRES(m) and MP-GMRES(m), if all algorithms can not solve the problem, there are almost no differences in the final attainable accuracy.

4.4.3 Analysis from the Total Number of Iterations

Experiments are conducted for each matrix using multiple m_{in} and ϵ_{in}. Table 4.2 and Table 4.3 provide the overall results on the total number of iterations. For FP64-/MP-IR using BiCGSTAB and FP64-/MP-GMRES(m) methods, the best results among all candidates of m_{in} and ϵ_{in} are listed. The number in parentheses is the value of m_{in} or ϵ_{in} that algorithms obtain the best results. For FP64 and MP-GMRES(m), m_{in} is used instead of m for the consistency with FP64-/MP-IR using BiCGSTAB.

First and foremost, if a problem can be solved by BiCGSTAB, MP-IR using BiCGSTAB can also solve it except `coupled`, `airfoil_2d`. This indicates that the problem-solving ability of BiCGSTAB and MP-IR using BiCGSTAB is almost the same. However, when comparing with GMRES(m) and MP-GMRES(m) method, it can be found from Figure 4.1 and Tables 4.2, 4.3 that GMRES(m) shows better robustness than BiCGSTAB and FP64-IR using BiCGSTAB, which means GMRES(m) can solve more problems, even for the matrics whose condition number is large. Comparison between MP-GMRES(m) and MP-IR using BiCGSTAB also shows the same tendency.

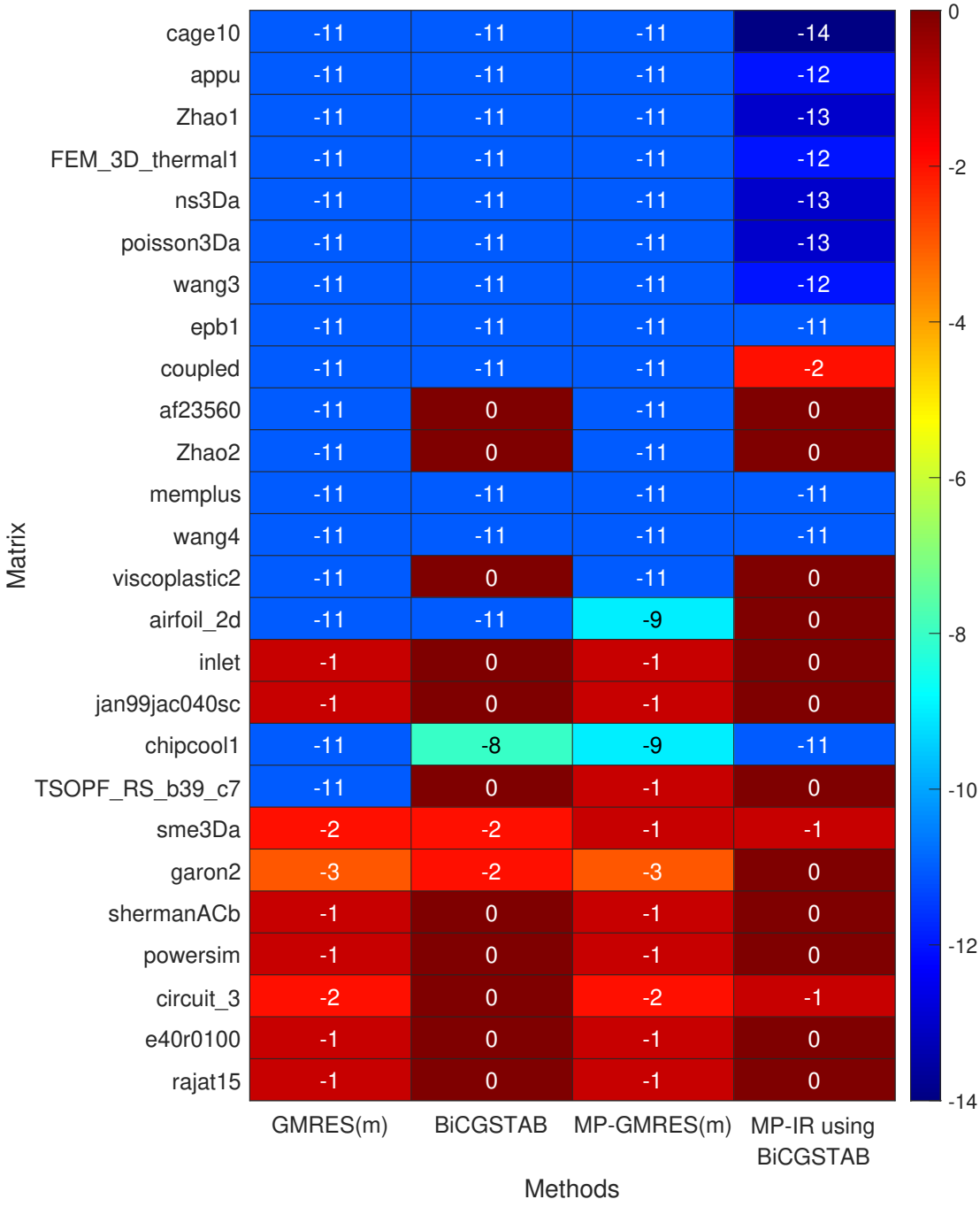


Figure 4.1: The maximum attainable accuracy of FP64-GMRES(m), MP-GMRES(m), BiCGSTAB and MP-IR using BiCGSTAB within the iteration limit: $\lfloor \log_{10}(\|b - Ax\|_2/\|b\|_2) \leq -11 \rfloor$ means it converged.

However, the BiCGSTAB (or FP64-IR using BiCGSTAB) requires less number of iterations than GMRES(m) method. This trend can also be found between MP-IR using BiCGSTAB and MP-GMRES(m). In particular, 7 out of 8 cases show this trend for matrices whose condition number is less than $O(10^3)$. This suggests that the employment of low precision BiCGSTAB into MP-IR is effective for matrices with small condition numbers. Thus, it is expected to benefit from MP-IR using BiCGSTAB if small condition numbers can be obtained by some techniques, such as preconditioning.

Another important point we can obtain from Table 4.3 is that FP64-IR using BiCGSTAB seems to be more sensitive to low precision computing (FP32) than GMRES(m) when the matrix condition number is large. For `coupled`, `memplus`, and `wang4`, there are some cases where FP64-IR using BiCGSTAB restarted by m_{in} or ϵ_{in} can converge, but MP-IR using BiCGSTAB restarted by m_{in} or ϵ_{in} cannot. On the contrary, both GMRES(m) and MP-GMRES(m) methods can converge.

Next, the comparison among BiCGSTAB, FP64-IR using BiCGSTAB and MP-IR using BiCGSTAB is given. Tables 4.4, 4.5 and 4.6 list the number of iterations of FP64-/MP-IR using BiCGSTAB which is stopped by m_{in} and ϵ_{in}, respectively. As the matrices with $\kappa_2(A) \geq O(10^7)$ are not solved by FP64-/MP-IR using BiCGSTAB, we do not list them in these three tables. In the above three tables, "—" represents that the method did not attain the convergence criterion.

Since the MP-IR using BiCGSTAB is different from BiCGSTAB in restart and low precision computing, the obtained results are analyzed from the following two aspects.

I. Impacts of restart and restart strategies(m_{in} and ϵ_{in}).

The impacts of restart are first analyzed by comparison between traditional BiCGSTAB and FP64-IR using BiCGSTAB. Next, the discussion on restart strategies is given by analyzing the results of FP64-/MP-IR using BiCGSTAB.

First, after introducing the restart, FP64-IR using BiCGSTAB requires almost the same number of iterations as BiCGSTAB does for most cases in Tables 4.2, 4.3. In Section 4.3.1, we theoretically analyzed the advantages and disadvantages of the restart, which may affect the convergence or increase the number of iterations. Cases that are consistent with this analysis can be found in the experiments. For example, the convergence rate is improved in the case of `memplus`, while the number of iterations increases for `airfoil_2d`. Besides, for `chipcool1`, the traditional BiCGSTAB method does not converge, but the FP64-IR using BiCGSTAB does.

Then, we discuss the impacts of m_{in} and ϵ_{in}. From Tables 4.2, 4.3, MP-IR using BiCGSTAB can solve more problems when stopped by m_{in}, e.g., `memplus`, `wang4`, `chipcool1`. However, when FP64-/MP-IR using BiCGSTAB stopped by both m_{in} and ϵ_{in} satisfy the convergence criterion, it is found that the same or less number of iterations are required when stopped by ϵ_{in}. The only exception is `chipcool1` in FP64-IR using BiCGSTAB.

However, the suitable restart frequencies for FP64-IR using BiCGSTAB and MP-IR using BiCGSTAB are different. Observed from Tables 4.4, 4.5 and 4.6, a small m_{in} is suitable for MP-IR while a large one is in FP64-IR. For ϵ_{in}, it is the opposite. A large ϵ_{in}, e.g., $\epsilon_{\text{in}} = 10^{-1}$ or 10^{-2}, is suitable for MP-IR while a small one is preferred in FP64-IR. These results are reasonable since the computations in MP-IR are performed in FP32.

II. Impacts of low precision computing.

By comparing the results between FP64-IR using BiCGSTAB and MP-IR using BiCGSTAB, it is clear that MP-IR using BiCGSTAB requires more number of iterations and it is consistent with previous theoretical analysis. This is also similar in GMRES(m) and MP-GMRES(m).

Besides, MP-IR using BiCGSTAB is more sensitive to m_{in} and ϵ_{in} than FP64-IR using BiCGSTAB. For example, the convergence behavior among different m_{in} or ϵ_{in} are quite different in MP-IR using BiCGSTAB for `memplus`. Only a few cases in MP-IR can converge while all cases in FP64 does. This indicates that after introducing low precision computing, MP-IR using BiCGSTAB has higher requirements for the choice of restart frequency. Detailed analysis of this case is given in Section 4.4.5 later.

4.4.4 Analysis from Execution Time

This section provides the analysis and comparison among traditional FP64-BiCGSTAB, MP-IR using BiCGSTAB, and MP-GMRES(m) in terms of the execution time.

Table 4.7 lists the execution time for attaining the convergence criterion. For MP-IR using BiCGSTAB and MP-GMRES(m), the best results among all candidates of m_{in} or ϵ_{in} are listed. Table 4.8 provides the estimated speedup calculated by the theoretical modeling in Section 4.3.2, where the number of iterations is the best result obtained in Table 4.2. The comparison of the speedups among BiCGSTAB, MP-IR using BiCGSTAB, and MP-GMRES(m) is listed in Table 4.9.

From Tables 4.7 and 4.9, it is found that MP-IR using BiCGSTAB is faster than traditional FP64-BiCGSTAB except for `ns3Da` and `poisson3Da`. In particular, a significant speedup is obtained in MP-IR using BiCGSTAB for `cage10`, `appu`, `Zhao1` and `FEM_3D_thermal1`. This indicates that applying low precision BiCGSTAB to MP-IR is helpful in improving the performance for problems with small condition numbers (around $O(10^3)$).

When comparing MP-IR using BiCGSTAB with MP-GMRES(m), MP-IR using BiCGSTAB outperforms MP-GMRES(m) for seven out of eight matrices, except for `ns3Da`. From Table 4.9, the actual speedups of MP-IR using BiCGSTAB over MP-GMRES(m) are basically consistent with the estimated speedups, which shows the potential of MP-IR using BiCGSTAB to have a better convergence rate than MP-GMRES(m) in solving practical problems with small condition numbers.

Table 4.2: The total number of inner iterations of matrices with small condition number ($\kappa_2(A) \leq O(10^3)$) for attaining the convergence criterion ($\|b - Ax\|_2/\|b\|_2 \leq 10^{-10}$): "—" represents that the method did not attain the criterion by the iteration limit, the number in parentheses is the value of m_{in} or ϵ_{in} that algorithms obtain the best results. (* means that there are multiple parameters that provide the best result; the smallest m_{in} or the largest ϵ_{in} is provided), the results on GMRES(m) are obtained from Tables 3.6 and 3.7.

$\kappa_2(A)$	Matrix Name	BiCGSTAB FP64	IR using BiCGSTAB FP64		IR using BiCGSTAB MP		GMRES(m) FP64	GMRES(m) MP
			m_{in}	ϵ_{in}	m_{in}	ϵ_{in}	m_{in}	m_{in}
$O(10^1)$	cage10	27	23 (50*)	17 (10^{-5})	60 (50*)	19 (10^{-2})	26 (50*)	59 (50)
$O(10^2)$	appu	76	73 (100*)	66 (10^{-5})	100 (50)	73 (10^{-1})	108 (200*)	114 (50)
	Zhao1	46	46 (50*)	31 (10^{-3})	150 (50)	32 (10^{-2})	43 (50*)	73 (50)
	FEM_3D_thermal1	184	170 (200*)	160 (10^{-2*})	200 (50*)	168 (10^{-2})	250 (300*)	270 (100)
	ns3Da	4356	5100 (300)	4088 (10^{-2})	5450 (50)	24963 (10^{-8})	1522 (500)	1522 (500)
$O(10^3)$	poisson3Da	131	131 (200*)	131 (10^{-10})	200 (50*)	184 (10^{-2})	184 (200*)	270 (100)
	wang3	221	224 (300*)	224 (10^{-10})	400 (100)	400 (10^{-2})	313 (400*)	587 (200)
	epb1	540	800 (400)	540 (10^{-10})	1000 (200)	937 (10^{-1})	1095 (500)	1153 (400)

Table 4.3: The total number of inner iterations of matrices with large condition number ($\kappa_2(A) \geq O(10^4)$) for attaining the convergence criterion ($\|b - Ax\|_2/\|b\|_2 \leq 10^{-10}$): "—" represents that the method did not attain the criterion by the iteration limit, the number in parentheses is the value of m_{in} or ϵ_{in} that algorithms obtain the best results. (* means that there are multiple parameters that provide the best result; the smallest m_{in} or the largest ϵ_{in} is provided), the results on GMRES(m) are obtained from Tables 3.6 and 3.7.

$\kappa_2(A)$	Matrix Name	BiCGSTAB FP64	IR using BiCGSTAB FP64		IR using BiCGSTAB MP		GMRES(m) FP64	GMRES(m) MP
			m_{in}	ϵ_{in}	m_{in}	ϵ_{in}	m_{in}	m_{in}
$O(10^4)$	coupled	2963	4500 (500)	2782 (10^{-5})	—	—	872 (500)	1091 (400)
	af23560	—	—	—	—	—	4305 (400)	4283 (400)
	Zhao2	—	—	—	—	—	2868 (300)	2846 (300)
$O(10^5)$	memplus	2513	1800 (200*)	1597 (10^{-1})	3800 (100)	—	1352 (500)	2044 (300)
	wang4	566	800 (200*)	588 (10^{-10})	1300 (100)	—	688 (500)	1423 (300)
	viscoplastic2	—	—	—	—	—	1299 (500)	1658 (200)
$O(10^6)$	airfoil_2d	2155	10400 (400)	3006 (10^{-5})	—	—	8172 (500)	—
	inlet	—	—	—	—	—	—	—
	jan99jac040sc	—	—	—	—	—	—	—
	chipcool1	—	10500 (300)	18486 (10^{-5})	16400 (200)	—	9636 (500)	—
$O(10^7)$	TSOPF_RS_b39_c7	—	—	—	—	—	392 (200)	—
	sme3Da	—	—	—	—	—	—	—
	garon2	—	—	—	—	—	—	—
	shermanACb	—	—	—	—	—	—	—
$O(10^8)$	powersim	—	—	—	—	—	—	—
	circuit_3	—	—	—	—	—	—	—
	e40r0100	—	—	—	—	—	—	—
	rajat15	—	—	—	—	—	—	—

As a remark, the case of `cage10` with m_{in} and case of `ns3Da` with ϵ_{in} do not converge in the experiments of execution time, which are different from the results of the number of iterations. We speculate that this might stem from the rounding errors caused by parallelization. For instance, changes in the order of computations could influence the convergence results.

4.4.5 Case Study

When low precision computing is introduced, the convergence behavior of FP64-IR using BiCGSTAB and MP-IR using BiCGSTAB show differences. Some typical and interesting cases are analyzed. In the figures, all the data plotted are computed with FP64 data and computations at each restart timing.

I. Cases where the matrix condition number is small($\kappa_2(A) \leq O(10^3)$): `epb1`, `ns3Da`

- **Case of `epb1` ($\kappa_2(A) = O(10^3)$)**

  Figure 4.2 provides the history of relative residual 2-norm for `epb1`. Compared with FP64-IR using BiCGSTAB, it is found that the decrease of relative residual 2-norm slows down in MP-IR using BiCGSTAB regardless of whether m_{in} or ϵ_{in} is used. This happens due to the use of low precision computing in the inner BiCGSTAB computations of MP-IR, which results in more number of iterations. And it is consistent with previous theoretical analyses.

  In the experiment on m_{in}, FP64-IR using BiCGSTAB tends to obtain better results with larger m_{in} while MP-IR using BiCGSTAB does not show this trend. In the experiment on ϵ_{in}, MP-IR using BiCGSTAB tends to obtain better results with larger ϵ_{in}, while FP64-IR using BiCGSTAB prefers smaller ϵ_{in}.

  Besides, an interesting phenomenon occurs in MP-IR using BiCGSTAB with ϵ_{in}. Seen from Figure 4.2d, when ϵ_{in} is less than 10^{-4}, the computed relative residual 2-norm obtained in the first inner BiCGSTAB loop can satisfy the convergence accuracy ϵ_{in}. However, the actual relative residual 2-norm computed in the outer loop using FP64 is different, only around 10^{-3}, which fails to achieve the corresponding ϵ_{in}. This happens due to the rounding errors. And it means that the first inner BiCGSTAB loop requires more number of iterations to achieve 10^{-3} as the ϵ_{in} becomes more and more smaller than 10^{-4}. A similar situation is not found in FP64-IR using BiCGSTAB.

- **Case of `wang3` ($\kappa_2(A) = O(10^3)$)**

  Figure 4.3 shows the history of relative residual 2-norm of `wang3`. From this case, it is clear to see the different impact of m_{in} and ϵ_{in} on FP64-/MP-IR using BiCGSTAB. For FP64-IR using BiCGSTAB, a large m_{in} or a small ϵ_{in} is prefer. On the contrary, a small m_{in} or a large ϵ_{in} is better for MP-IR

using BiCGSTAB. Additionally, after the first inner BiCGSTAB loop, the actual relative residual 2-norm in FP64 also fails to attain the given ϵ_{in} when $\epsilon_{\text{in}} \leq 10^{-5}$, which also occurs in the case of `epb1`.

- **Case of `ns3Da` ($\kappa_2(A) = O(10^2)$)**

  The numerical behavior for `ns3Da` is quite different from other matrices whose condition number is less than $O(10^3)$. The history of relative residual 2-norm is shown in Figure 4.4. In the experiments on m_{in}, there are some cases in which the relative residual 2-norm does not decrease as the iteration increases. And MP-IR using BiCGSTAB fails to converge in some of these cases due to the appearance of NaN. As for the experiments on ϵ_{in}, despite the fact that the relative residual 2-norm decreases as iteration increases, MP-IR using BiCGSTAB still fails to converge except for the case where $\epsilon_{\text{in}} = 10^{-8}$. By checking the computed relative residual 2-norm in the inner BiCGSTAB, it is found that NaN appears, thus resulting in the breakdown. The detailed mechanism causing these results is not clear and needs further study.

  Besides, the case of `ns3Da` is also analyzed in the research on GMRES(m). Both GMRES(m) and MP-GMRES(m) show better convergence behavior than FP64-/MP-IR using BiCGSTAB, which reflects that BiCGSTAB is more sensitive to a target problem than GMRES.

II. Cases where the matrix condition number is large($\kappa_2(A) \geq O(10^4)$): `memplus`, `chipcool1`

- **Case of `memplus` ($\kappa_2(A) = O(10^5)$)**

  The history of the relative residual 2-norm is shown in Figure 4.5. From the figure, FP64-IR using BiCGSTAB can converge with each m_{in} and ϵ_{in}. However, only when $m_{\text{in}} = 50, 100$, MP-IR using BiCGSTAB can converge, while it breaks down in other situations due to `NaN`. This shows the great difference in the choice of m_{in} and ϵ_{in} between FP64-IR and MP-IR using BiCGSTAB, and MP-IR using BiCGSTAB has a higher demand for the choices.

  To better clarify the convergence behavior, the computed relative residual 2-norm obtained in the inner BiCGSTAB loop is provided. For MP-IR using BiCGSTAB, it breaks down in the first inner BiCGSTAB loop when $\epsilon_{\text{in}} = 10^{-2}$, and Figure 4.6 provides the detailed convergence process. The data plotted in the figure is computed by the formula in Line 11 of Algorithm 5. Probably due to the influence of rounding error, the history of relative residual 2-norm shows an increasing trend. MP-IR using BiCGSTAB then fails to converge because of `NaN`. This case again confirms that BiCGSTAB is very sensitive to rounding errors; the change in accuracy or parameter value may lead to the breakdown, especially when dealing with problems with large condition numbers. Under this

situation, some techniques such as preconditioning may be necessary to apply MP-IR using BiCGSTAB to practical problems.

- **Case of `chipcool1` ($\kappa_2(A) = O(10^6)$)**

  For `chipcool1`, the most significant behavior is that the traditional FP64-BiCGSTAB does not converge while both FP64-IR using BiCGSTAB and MP-IR using BiCGSTAB converge. The history of relative residual 2-norm, including FP64-BiCGSTAB, FP64-IR using BiCGSTAB, and MP-IR using BiCGSTAB is shown in Figure 4.7. The decrease of the relative residual 2-norm in FP64-BiCGSTAB is slow; sometimes, there is a drastic increase or stagnation probably due to the rounding errors. Therefore, although the relative residual 2-norm is decreasing overall, FP64-BiCGSTAB can not converge within the iteration limit. However, introducing a restart improves this phenomenon in FP64-/MP-IR using BiCGSTAB. The convergence rate is improved and becomes faster; therefore, FP64-IR using BiCGSTAB and MP-IR using BiCGSTAB can converge in some cases. And the sharp increase in FP64-BiCGSTAB also brings difficulties for the convergence of MP-IR using BiCGSTAB with ϵ_{in}.

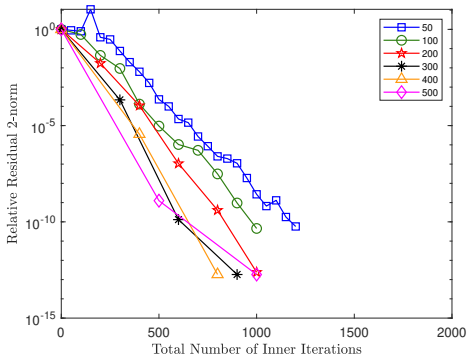
4.5 Conclusion

We apply BiCGSTAB with low precision computing (FP32) as an inner solver of mixed precision iterative refinement(MP-IR), which can combine high precision and low precision and provide the solution with the same accuracy as that by the method using only FP64, and propose a MP-IR using BiCGSTAB algorithm. To clarify the performance and application prospects of the MP algorithm, we investigate its numerical behavior from both theoretical and practical perspectives and compared it with the MP-GMRES(m) method.

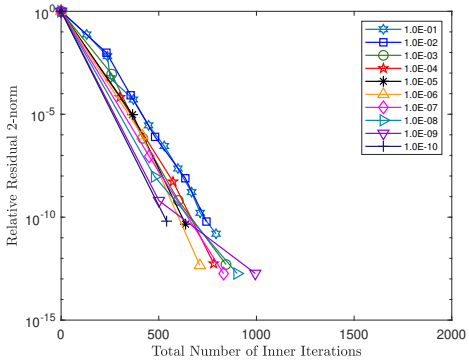
Using the same matrix dataset as the research on MP-GMRES(m), a series of comprehensive experiments are conducted on a CPU platform, and the obtained results are analyzed and compared with MP-GMRES(m) from the viewpoints of the maximum attainable accuracy, number of iterations, and execution time. Since a restart and low precision computing are introduced in the MP-IR using BiCGSTAB, the impact of these two factors is also discussed.

By comparing the results of traditional BiCGSTAB, FP64-IR using BiCGSTAB, and MP-IR using BiCGSTAB, we can observe the following results:

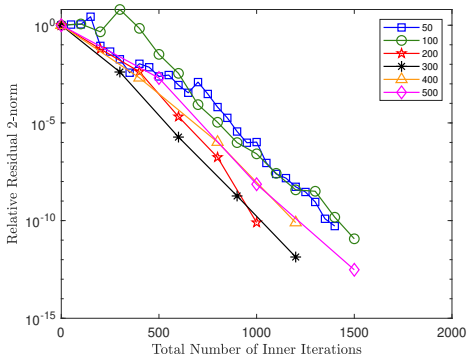
- The MP-IR using BiCGSTAB can provide a solution with the same accuracy as those by traditional BiCGSTAB using only FP64.
- The MP-IR using BiCGSTAB is faster than BiCGSTAB for most cases, especially for the problems with small condition numbers.
- By introducing restart, FP64-/MP-IR using BiCGSTAB has the possibility to solve the problems that BiCGSTAB cannot solve, for example, the case of `chipcool1`.



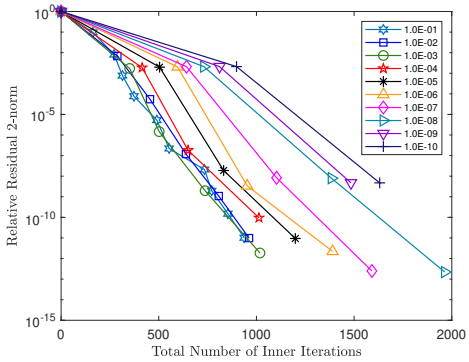
(a) FP64-IR using BiCGSTAB with different m_{in}



(b) FP64-IR using BiCGSTAB with different ϵ_{in}

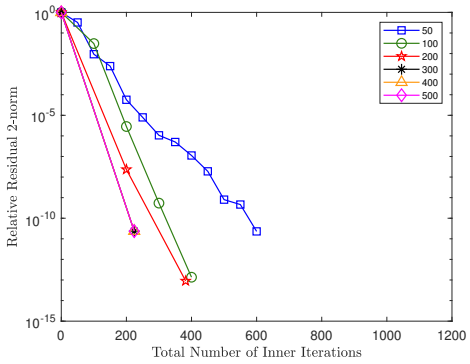


(c) MP-IR using BiCGSTAB with different m_{in}

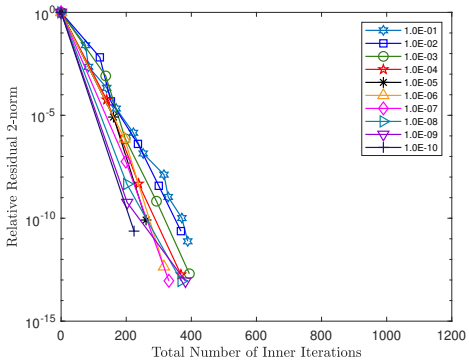


(d) MP-IR using BiCGSTAB with different ϵ_{in}

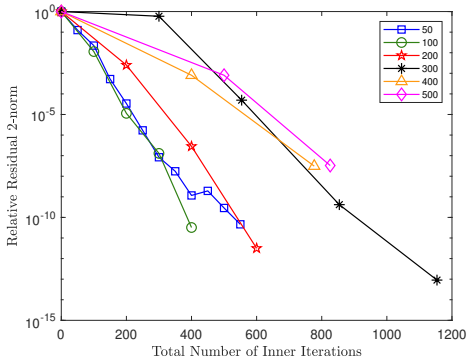
Figure 4.2: Relative residual 2-norm history for `epb1` with different m_{in} and ϵ_{in}.



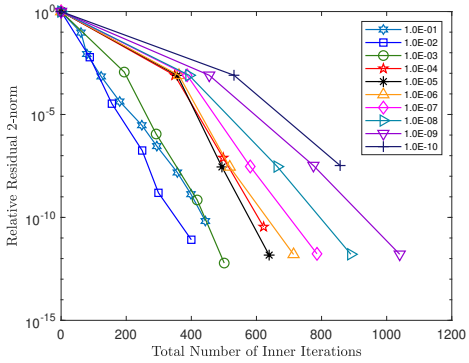
(a) FP64-IR using BiCGSTAB with different m_{in}



(b) FP64-IR using BiCGSTAB with different ϵ_{in}

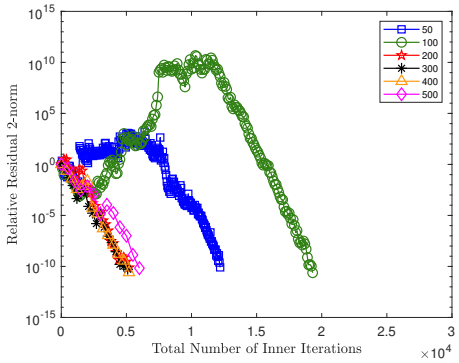


(c) MP-IR using BiCGSTAB with different m_{in}

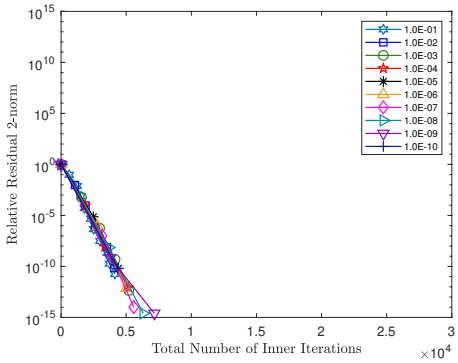


(d) MP-IR using BiCGSTAB with different ϵ_{in}

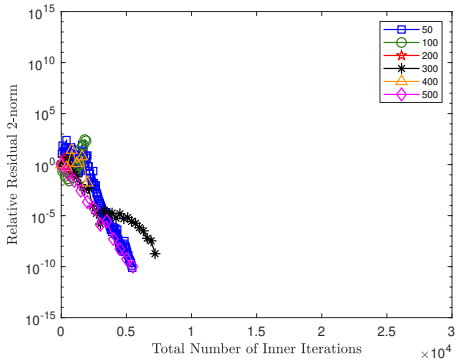
Figure 4.3: Relative residual 2-norm history for `wang3` with different m_{in} and ϵ_{in}: data is plotted halfway due to the breakdown of the method in some cases.



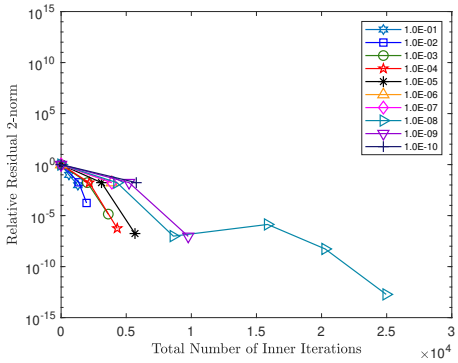
(a) FP64-IR using BiCGSTAB with different m_{in}



(b) FP64-IR using BiCGSTAB with different ϵ_{in}

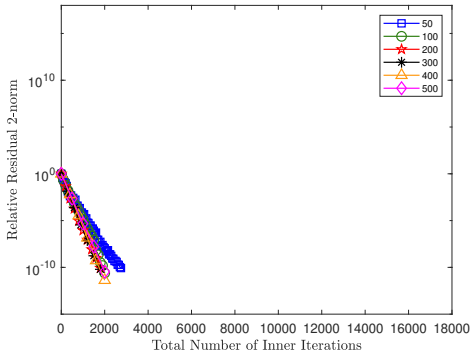


(c) MP-IR using BiCGSTAB with different m_{in}

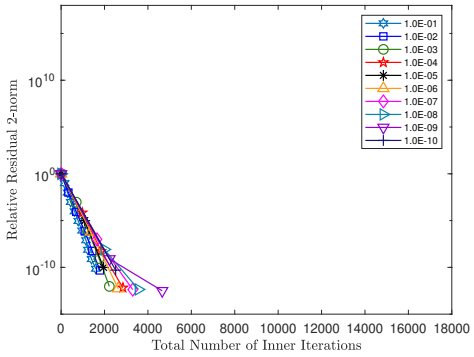


(d) MP-IR using BiCGSTAB with different ϵ_{in}

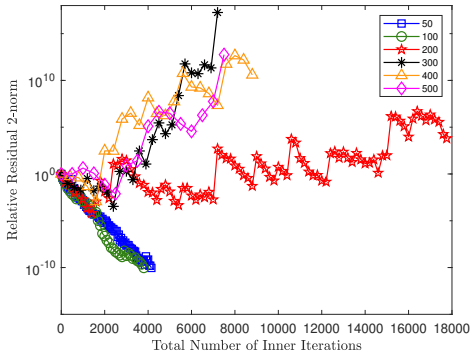
Figure 4.4: Relative residual 2-norm history for `ns3Da` with different m_{in} and ϵ_{in} : data is plotted halfway due to the breakdown of the method in some cases.



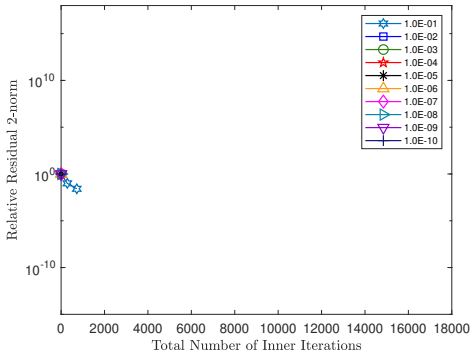
(a) FP64-IR using BiCGSTAB with different m_{in}



(b) FP64-IR using BiCGSTAB with different ϵ_{in}



(c) MP-IR using BiCGSTAB with different m_{in}



(d) MP-IR using BiCGSTAB with different ϵ_{in}

Figure 4.5: Relative residual 2-norm history for `memplus` with different m_{in} and ϵ_{in} : data is plotted halfway due to the breakdown of the method in some cases.

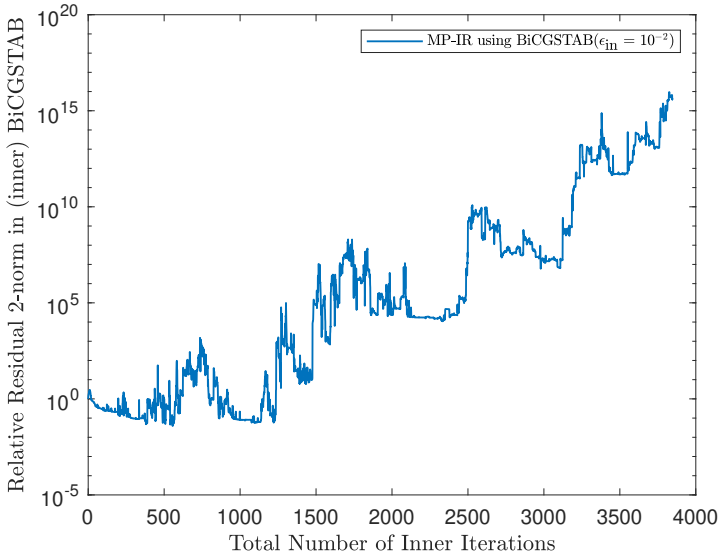


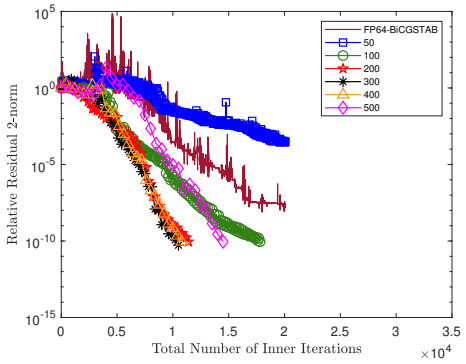
Figure 4.6: Relative residual 2-norm history of the first inner BiCGSTAB loop in MP-IR using BiCGSTAB for `memplus`: the plotted data is computed by the recurrence formula (Line 11 of Algorithm 5) in FP32-BiCGSTAB.

- For the choice of m_{in} and ϵ_{in}, IR using BiCGSTAB prefers a large m_{in} and a small ϵ_{in}, while the MP-IR using BiCGSTAB favors a small m_{in} and a large ϵ_{in}.

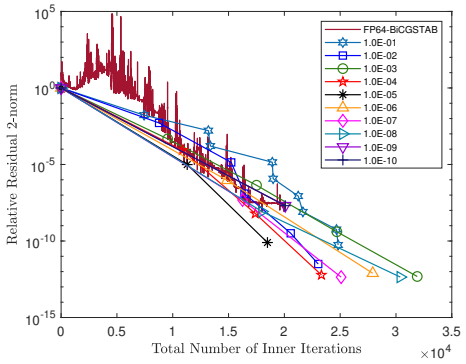
Besides, the following observations are found by comparing MP-IR using BiCGSTAB and MP-GMRES(m):

- The MP-GMRES(m) has better robustness than MP-IR using BiCGSTAB.
- The MP-IR using BiCGSTAB sometimes outperforms MP-GMRES(m) in both the number of iterations and execution time, especially for problems whose condition number is small.

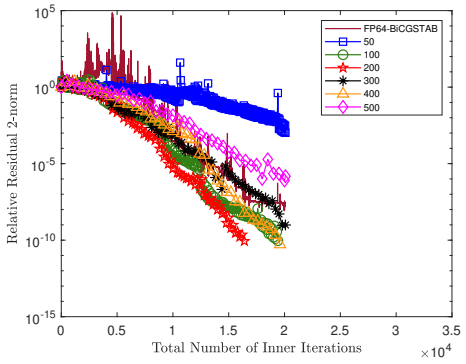
As one of the possible improvements for robustness, some preconditioners, which can be used to get a sufficiently small condition number, are expected to benefit the problem-solving ability of MP-IR using BiCGSTAB. Through this research, MP-IR using BiCGSTAB shows a strong potential for its implementation into practical applications and is expected to obtain good results.



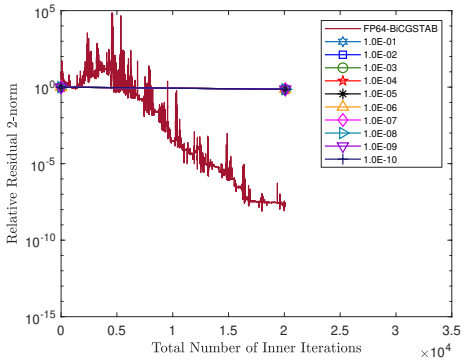
(a) BiCGSTAB and FP64-IR using BiCGSTAB



(b) BiCGSTAB and FP64-IR using BiCGSTAB



(c) BiCGSTAB and MP-IR using BiCGSTAB



(d) BiCGSTAB and MP-IR using BiCGSTAB

Figure 4.7: Relative residual 2-norm history for `chipcool1` : including FP64-BiCGSTAB and FP64-/MP-IR using BiCGSTAB with different m_{in} and ϵ_{in}, data is plotted halfway due to the breakdown of the method in some cases.

Table 4.4: The total number of inner iterations for attaining the convergence criterion in FP64-/MP-IR using BiCGSTAB stopped by m_{in}: $\epsilon_{\text{in}} = 10^{-10}$, "—" represents that the method did not attain the criterion by the iteration limit.

$\kappa_2(A)$	Matrix Name	Precision	m_{in} in IR using BiCGSTAB 50	100	200	300	400	500
$O(10^1)$	cage10	FP64	23	23	23	23	23	23
		MP	60	60	60	60	60	60
$O(10^2)$	appu	FP64	100	73	73	73	73	73
		MP	100	144	144	144	144	144
	Zhao1	FP64	46	46	46	46	46	46
		MP	150	219	—	—	1258	685
	FEM_3D_thermal1	FP64	200	200	170	170	170	170
		MP	200	200	372	423	423	423
	ns3Da	FP64	12200	19300	5200	5100	5200	6000
		MP	5450	—	—	—	—	5500
$O(10^3)$	poisson3Da	FP64	200	200	131	131	131	131
		MP	200	200	264	264	264	264
	wang3	FP64	600	400	382	224	224	224
		MP	550	400	600	1154	—	—
	epb1	FP64	1200	1000	1000	900	800	1000
		MP	1400	1500	1000	1200	1200	1500
$O(10^4)$	coupled	FP64	—	—	—	6000	4800	4500
		MP	—	—	—	—	—	—
	af23560	FP64	—	—	—	—	—	—
		MP	—	—	—	—	—	—
	Zhao2	FP64	—	—	—	—	—	—
		MP	—	—	—	—	—	—
$O(10^5)$	memplus	FP64	2750	2000	1800	1800	2000	2000
		MP	4150	3800	—	—	—	—
	wang4	FP64	2100	1700	800	900	800	939
		MP	1500	1300	1400	1500	—	—
	viscoplastic2	FP64	—	—	—	—	—	—
		MP	—	—	—	—	—	—
$O(10^6)$	airfoil_2d	FP64	—	—	—	10500	10400	13000
		MP	—	—	—	—	—	—
	inlet	FP64	—	—	—	—	—	—
		MP	—	—	—	—	—	—
	jan99jac040sc	FP64	—	—	—	—	—	—
		MP	—	—	—	—	—	—
	chipcool1	FP64	—	17800	11400	10500	10800	14500
		MP	—	19400	16400	—	19600	—

Table 4.5: The total number of inner iterations for attaining the convergence criterion in FP64-IR using BiCGSTAB stopped by ϵ_{in}: $m_{\text{in}} = n$, "—" represents that the method did not attain the criterion by the iteration limit.

$\kappa_2(A)$	Matrix Name	ϵ_{in} in FP64-IR using BiCGSTAB									
		10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}
$O(10^1)$	cage10	23	35	30	26	22	17	22	22	18	20
$O(10^2)$	appu	73	126	114	96	79	66	78	79	67	78
	Zhao1	46	87	69	54	40	35	41	31	33	36
	FEM_3D_thermal1	170	329	284	256	204	160	204	199	160	173
	ns3Da	4356	7163	6304	5586	5030	4390	5181	5200	4088	4128
$O(10^3)$	poisson3Da	131	225	210	190	161	145	188	210	166	210
	wang3	224	382	369	331	316	261	368	394	368	389
	epb1	540	994	901	832	710	637	781	844	744	793
$O(10^4)$	coupled	2930	5562	4530	4046	3595	2782	3979	4188	3309	4113
	af23560	—	—	—	—	—	—	—	—	—	—
	Zhao2	—	—	—	—	—	—	—	—	—	—
$O(10^5)$	memplus	2513	4658	3564	3300	2630	1956	2847	2223	1794	1597
	wang4	588	987	891	659	656	594	720	818	792	806
	viscoplastic2	—	—	—	—	—	—	—	—	—	—
$O(10^6)$	airfoil_2d	4978	4639	6036	—	3198	3006	3955	3437	4507	—
	inlet	—	—	—	—	—	—	—	—	—	—
	jan99jac040sc	—	—	—	—	—	—	—	—	—	—
	chipcool1	—	—	30388	25087	27917	18486	23322	31904	23035	24816

Table 4.6: The total number of inner iterations for attaining the convergence criterion in MP-IR using BiCGSTAB stopped by ϵ_{in}: $m_{\text{in}} = n$, "—" represents that the method did not attain the criterion by the iteration limit.

$\kappa_2(A)$	Matrix Name	ϵ_{in} in MP-IR using BiCGSTAB									
		10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}
$O(10^1)$	cage10	60	56	53	44	39	35	37	22	19	20
$O(10^2)$	appu	144	134	115	111	95	86	94	87	74	73
	Zhao1	925	892	3896	657	59	48	48	36	32	36
	FEM_3D_thermal1	423	385	329	290	251	200	233	219	168	176
	ns3Da	—	—	24963	—	—	—	—	—	—	—
$O(10^3)$	poisson3Da	264	240	219	200	272	224	198	205	184	228
	wang3	—	1040	890	786	714	639	622	501	400	443
	epb1	—	—	1964	1591	1390	1199	1013	1018	962	937
$O(10^4)$	coupled	—	—	—	—	—	—	—	—	—	—
	af23560	—	—	—	—	—	—	—	—	—	—
	Zhao2	—	—	—	—	—	—	—	—	—	—
$O(10^5)$	memplus	—	—	—	—	—	—	—	—	—	—
	wang4	—	—	—	—	—	—	—	—	—	—
	viscoplastic2	—	—	—	—	—	—	—	—	—	—
$O(10^6)$	airfoil_2d	—	—	—	—	—	—	—	—	—	—
	inlet	—	—	—	—	—	—	—	—	—	—
	jan99jac040sc	—	—	—	—	—	—	—	—	—	—
	chipcool1	—	—	—	—	—	—	—	—	—	—

Table 4.7: The execution time for attaining the convergence criterion ($\|b - Ax\|_2/\|b\|_2 \leq 10^{-10}$): "—" represents that the method did not attain the criterion by the iteration limit, for MP-IR using BiCGSTAB and MP-GMRES(m), the best result among all candidates of m_{in} or ϵ_{in} is listed.

$\kappa_2(A)$	Matrix Name	BiCGSTAB FP64	IR using BiCGSTAB MP		GMRES(m) MP
			m_{in}	ϵ_{in}	m_{in}
$O(10^1)$	cage10	1.08×10^{-2}	—	1.76×10^{-3}	1.43×10^{-2}
$O(10^2)$	appu	3.86×10^{-2}	3.47×10^{-2}	2.17×10^{-2}	4.58×10^{-2}
	Zhao1	2.26×10^{-2}	4.22×10^{-2}	7.55×10^{-3}	2.90×10^{-2}
	FEM_3D_thermal1	6.07×10^{-2}	4.36×10^{-2}	2.45×10^{-2}	6.36×10^{-2}
	ns3Da	1.73×10^{0}	2.80×10^{0}	—	9.59×10^{-1}
$O(10^3)$	poisson3Da	2.93×10^{-2}	4.45×10^{-2}	3.74×10^{-2}	8.32×10^{-2}
	wang3	8.95×10^{-2}	8.52×10^{-2}	8.50×10^{-2}	2.35×10^{-1}
	epb1	9.78×10^{-2}	8.93×10^{-2}	8.19×10^{-2}	3.24×10^{-1}

Table 4.8: The estimated speedup of MP-IR using BiCGSTAB and MP-GMRES(m): The estimated speedup is obtained by the estimated model, and $T_{\text{mp-ir-bicgstab}}$ and $T_{\text{mp-gmres}}$ mean the estimated computation time of MP-IR using BiCGSTAB and that of MP-GMRES(m), respectively. # of iter. represents the total number of iterations.

$\kappa_2(A)$	Matrix Name	γ	MP-IR using BiCGSTAB # of iter.	MP-GMRES(m) # of iter.	m	$\frac{M_{\text{bicgstab}}^{(\text{FP32})}}{M_{\text{m-gmres}}^{(\text{FP32})}}$	$\frac{T_{\text{mp-ir-bicgstab}}}{T_{\text{mp-gmres}}}$
$O(10^1)$	cage10	13	19	59	50	1.07	0.35
$O(10^2)$	appu	132	73	114	50	1.83	1.17
	Zhao1	5	32	73	50	0.68	0.30
	FEM_3D_thermal1	24	168	270	100	1.01	0.63
	ns3Da	82	5450	1522	500	0.80	2.87
$O(10^3)$	poisson3Da	26	184	270	100	1.05	0.71
	wang3	7	400	587	200	0.29	0.20
	epb1	6	937	1153	400	0.14	0.11

Table 4.9: The Comparison of estimated and actual speedup among BiCGSTAB, MP-IR using BiCGSTAB and MP-GMRES(m): $T_{\text{mp-ir-bicgstab}}$ and $T_{\text{mp-gmres}}$ mean the estimated computation time of MP-IR using BiCGSTAB and that of MP-GMRES(m), respectively; $T^{(\text{actual})}_{\text{bicgstab}}$, $T^{(\text{actual})}_{\text{mp-ir-bicgstab}}$ and $T^{(\text{actual})}_{\text{mp-gmres}}$ mean the execution time of BiCGSTAB, MP-IR using BiCGSTAB (the best among two results) and MP-GMRES(m), respectively.

$\kappa_2(A)$	Matrix Name	$\frac{T_{\text{mp-ir-bicgstab}}}{T_{\text{mp-gmres}}}$	$\frac{T^{(\text{actual})}_{\text{mp-ir-bicgstab}}}{T^{(\text{actual})}_{\text{mp-gmres}}}$	$\frac{T^{(\text{actual})}_{\text{mp-ir-bicgstab}}}{T^{(\text{actual})}_{\text{bicgstab}}}$
$O(10^1)$	cage10	0.35	0.12	0.16
$O(10^2)$	appu	1.17	0.47	0.56
	Zhao1	0.30	0.26	0.33
	FEM_3D_thermal1	0.63	0.39	0.40
	ns3Da	2.87	2.91	1.62
$O(10^3)$	poisson3Da	0.71	0.45	1.27
	wang3	0.20	0.36	0.95
	epb1	0.11	0.25	0.84

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The research focuses on mixed precision computing and combining it with Krylov subspace methods. In this research, I consider the GMRES(m) method and the BiCGSTAB method, which are typical Krylov subspace methods for sparse and non-symmetric linear systems. By exploiting the high ability of low precision computing, I investigate different mixed precision algorithms for solving a linear system $Ax = b$, where A is large, sparse, and non-symmetric, and aim to achieve high performance and accelerate various practical applications. Basically, I have three contributions.

1. Investigate the characteristics and effectiveness of MP-GMRES(m).

   In Chapter 3, I studied the traditional GMRES(m) method and investigated the characteristics and effectiveness of its mixed precision variant, namely MP-GMRES(m). In this chapter, the traditional GMRES(m) and MP-GMRES(m) methods are studied from both theoretical and practical aspects, and I implemented them on a standard CPU platform. Comprehensive experiments on various test matrices with multiple sets of parameter configurations are conducted. Detailed analysis and comparison between GMRES(m) and MP-GMRES(m) are given from the aspects of the maximum attainable accuracy, the number of iterations, and the execution time.

2. Propose MP-IR using BiCGSTAB and investigate its characteristics and effectiveness.

   In Chapter 4, I proposed a new mixed precision algorithm, namely MP-IR using BiCGSTAB, based on BiCGSTAB and mixed precision iterative refinement. In this chapter, I first introduce the BiCGSTAB method. Then, I explain how to introduce low precision computing into BiCGSTAB and implement the developed mixed precision algorithm on a CPU platform. Similar to the research on MP-GMRES(m), the algorithms are

studied and analyzed from both theoretical and practical perspectives, and comprehensive numerical results are analyzed from three aspects. Detailed comparisons with the traditional BiCGSTAB method, as well as MP-GMRES(m), are also provided.

3. Clarify the applicability of mixed precision computing in numerical linear solvers.

   In my research, the MP-GMRES(m) method and the MP-IR using BiCGSTAB method for sparse and non-symmetric linear systems are investigated. The obtained results clarify the applicability of mixed precision computing in numerical linear solvers, which may achieve high performance and accelerate a variety of practical applications requiring linear solvers. In addition, reducing the time of the simulation program will also significantly impact data science applications because simulation is now one of the important data generators.

Detailed analysis and comparisons are made from three aspects: the maximum attainable accuracy, the number of iterations, and the execution time. The main conclusions are listed as follows.

1. Problem solving ability.

   It is observed that MP-GMRES(m) has almost the same problem-solving ability as GMRES(m). The same tendency is also found between MP-IR using BiCGSTAB and BiCGSTAB, especially for matrices with small condition numbers. When all methods can not solve the problem, the final attainable accuracy is almost no different. However, MP-GMRES(m) can solve more problems than MP-IR using BiCGSTAB.

2. In the aspect of the number of iterations.

   After employing mixed precision computing, MP-GMRES(m) has almost the same or more iterations than GMRES(m). As for MP-IR using BiCGSTAB and FP64-IR using BiCGSTAB, it shows the same trend. When comparing between the two mixed precision algorithms, MP-IR using BiCGSTAB requires less number of iterations than MP-GMRES(m) method for most cases.

3. In the aspect of execution time.

   After introducing low precision computing, both MP-GMRES(m) and MP-IR using BiCGSTAB are able to get improvement in execution time. MP-IR using BiCGSTAB is, although sensitive to problems, the faster among the four methods for some problems.

5.2 Future Work

Mixed Precision computing is one of the most recent research topics in numerical linear algebra and high-performance computing. Not only the theoretical

and mathematical studies of the algorithms but also the practical studies in applications are important.

First, I consider extending my research to the applications in other fields, such as data science and AI. I plan to implement the mixed precision algorithms developed in my research as numerical libraries for some hardware, such as State of the art GPUs, so as to be easily and better used in various practical application programs. In practice, applications in fields such as data science and AI usually require fast linear algebra computations, which my research is expected to improve and accelerate. At the same time, some techniques in data science and AI are also expected to be combined with my research to better solve the problems of algorithm selection for different problems.

Secondly, from the optimization of the developed mixed precision algorithm itself, there are also some details in my research that need further study.

1. Employ a preconditioner into MP-IR using BiCGSTAB algorithm.

   From the results obtained from this study, a problem with a small condition number tends to be well solved by MP-IR using BiCGSTAB. Thus, reducing the condition number by a suitable preconditioner is expected to achieve better results in the MP-IR using BiCGSTAB algorithm. Besides, although some studies on MP-GMRES(m) using preconditioners have been reported, its numerical behavior is still worth to be studied more specifically. Therefore, further studies of the numerical behavior of MP-IR using BiCGSTAB and MP-GMRES(m) using a preconditioner are necessary.

2. Implementation of MP-IR using BiCGSTAB on GPU platform.

   In this study, MP-IR using BiCGSTAB is investigated on a CPU platform, but its performance on a GPU platform has not been evaluated yet. By taking full advantage of the GPU, evaluating the performance of MP-IR using BiCGSTAB on GPU is valuable for further optimization.

3. More study of restart frequency (m_{in} and ϵ_{in}).

   From the studies of other scholars and our obtained results, the restart frequency impacts the performance of MP-GMRES(m) and MP-IR using BiCGSTAB. Its detailed effects are worth further study, which not only benefits MP-GMRES(m) and MP-IR using BiCGSTAB but also benefits other Krylov subspace methods. Besides, some ideas and attempts for restart frequency are studied in GMRES(m) [71, 7, 34, 35]. It is also a worthwhile direction to exploit these ideas into mixed precision algorithms, i.e., dynamically adjust the restart frequency according to the convergence process of the problem.

Bibliography

[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] Bai, Z.-Z. and Yin, J.-F.: Modified incomplete orthogonal factorization methods using Givens rotations, Computing, Vol. 86, pp. 53–69 (2009).

[7] 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).

[8] Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. and der Vorst, H. V.: Templates for the

Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, PA (1994).

[9] 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).

[10] 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 Transactions on Mathematical Software, Vol. 34, No. 4, pp. 17:1–17:22 (2008).

[11] 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).

[12] 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).

[13] 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).

[14] Cipra, B. A.: The Best of the 20th Century: Editors Name Top 10 Algorithms, SIAM News, Vol. 33, No. 4.

[15] Clark, M. A., Babich, R., Barros, K., Brower, R. C. and Rebbi, C.: Solving lattice QCD systems of equations using mixed precision solvers on GPUs, Computer Physics Communications, Vol. 181, No. 9, pp. 1517–1528 (2010).

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

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

[18] der Vorst, H. A. V.: Iterative Krylov methods for large linear systems, No. 13, Cambridge University Press (2003).

[19] Dongarra, J. J., Duff, L. S., Sorensen, D. C. and Vorst, H. A. V.: Numerical Linear Algebra for High Performance Computers, SIAM (1998).

[20] Fletcher, R.: Conjugate gradient methods for indefinite systems, Numerical Analysis (Watson, G. A., ed.), Lecture Notes in Mathematics, Springer, pp. 73–89 (1976).

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

[22] Golub, G. H. and Van Loan, C. F.: Matrix computations, JHU press (2013).

[23] GUPTA, G.: What's the Difference Between Single-, Double-, Multi- and Mixed-Precision Computing? 2019, Nov 15.

[24] Gupta, S., Agrawal, A., Gopalakrishnan, K. and Narayanan, P.: Deep Learning with Limited Numerical Precision, Proceedings of the 32nd International Conference on Machine Learning (Bach, F. and Blei, D., eds.), Proceedings of Machine Learning Research, Vol. 37, Lille, France, PMLR, pp. 1737–1746 (2015).

[25] Göbel, F., Grützmacher, T., Ribizel, T. and Anzt, H.: Mixed Precision Incomplete and Factorized Sparse Approximate Inverse Preconditioning on GPUs, Euro-Par 2021: Parallel Processing (Sousa, L., Roma, N. and Tomás, P., eds.), Lecture Notes in Computer Science, Springer International Publishing, pp. 550–564 (2021).

[26] 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).

[27] Hestenes, M. R. and Stiefel, E.: Methods of conjugate gradients for solving linear systems, Journal of research of the National Bureau of Standards, Vol. 49, pp. 409–435 (1952).

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

[29] Higham, N. J. and Mary, T.: Mixed precision algorithms in numerical linear algebra, Acta Numerica, Vol. 31, pp. 347–414 (2022).

[30] 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).

[31] 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).

[32] Ichimura, T., Fujita, K., Yamaguchi, T., Naruse, A., Wells, J. C., Schulthess, T. C., Straatsma, T. P., Zimmer, C. J., Martinasso, M., Nakajima, K., Hori, M. and Maddegedara, L.: A Fast Scalable Implicit Solver for Nonlinear Time-Evolution Earthquake City Problem on Low-Ordered Unstructured Finite Elements with Artificial Intelligence and Transprecision Computing, SC18:

International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 627–637 (2018).

[33] 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 Transactions on Magnetics, Vol. 48, No. 2, pp. 467–470 (2012).

[34] 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).

[35] 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).

[36] J.H.WILKINSON: Rounding Errors in Algebraic Processes, Prentice Hall, Englewood Cliffs, N.J. (1963).

[37] Kolotilina, L. Y. and Yeremin, A. Y.: Factorized Sparse Approximate Inverse Preconditionings I. Theory, SIAM Journal on Matrix Analysis and Applications, Vol. 14, No. 1, pp. 45–58 (1993).

[38] Krylov, A.: De la résolution numérique de l'équation servant à déterminer dans des questions de mécanique appliquée les fréquences de petites oscillations des systèmes matériels, Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na, pp. 491–539 (1931).

[39] Kuchaiev, O., Ginsburg, B., Gitman, I., Lavrukhin, V., Li, J., Nguyen, H., Case, C. and Micikevicius, P.: Mixed-precision training for NLP and speech recognition with openseq2seq, arXiv preprint arXiv:1805.10387 (2018).

[40] 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).

[41] Kurth, T., Treichler, S., Romero, J., Mudigonda, M., Luehr, N., Phillips, E., Mahesh, A., Matheson, M., Deslippe, J., Fatica, M., Prabhat, P. and Houston, M.: Exascale Deep Learning for Climate Analytics, SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 649–660 (2018).

[42] 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: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, pp. 50–50 (2006).

[43] Liesen, J. and Tichỳ, P.: Convergence analysis of Krylov subspace methods, GAMM-Mitteilungen, Vol. 27, No. 2, pp. 153–173 (2004).

[44] 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).

[45] Lindquist, N., Luszczek, P. and Dongarra, J.: Accelerating Restarted GMRES with Mixed Precision Arithmetic, IEEE Transactions on Parallel and Distributed Systems, Vol. 33, pp. 1027–1037 (2022).

[46] 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).

[47] Micikevicius, P.: Mixed-Precision Training of Deep Neural Networks. 2017, Oct 11.

[48] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G. et al.: Mixed precision training, arXiv preprint arXiv:1710.03740 (2017).

[49] Mittal, R. and Al-Kurdi, A.: An efficient method for constructing an ILU preconditioner for solving large sparse nonsymmetric linear systems by the GMRES method, Computers & Mathematics with Applications, Vol. 45, No. 10, pp. 1757–1772 (2003).

[50] Moler, C. B.: Iterative Refinement in Floating Point, Journal of the Association for Computing Machinery(ACM), Vol. 14, No. 2, pp. 316–321 (1967).

[51] 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).

[52] 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).

[53] 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).

[54] Ootomo, H. and Yokota, R.: Recovering single precision accuracy from Tensor Cores while surpassing the FP32 theoretical peak performance, The International Journal of High Performance Computing Applications, Vol. 36, No. 4, pp. 475–491 (2022).

[55] Peng, H., Wu, K., Wei, Y., Zhao, G., Yang, Y., Liu, Z., Xiong, Y., Yang, Z., Ni, B., Hu, J. et al.: FP8-LM: Training FP8 Large Language Models, arXiv preprint arXiv:2310.18313 (2023).

[56] Saad, Y.: Krylov Subspace Methods on Supercomputers, SIAM Journal on Scientific and Statistical Computing, Vol. 10, No. 6, pp. 1200–1232 (1989).

[57] 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).

[58] Saad, Y.: Iterative Methods for Sparse Linear Systems, Society for Industrial and Applied Mathematics (2003).

[59] Singh, R. and Singh, K. M.: On preconditioned BiCGSTAB solver for MLPG method applied to heat conduction in 3D complex geometry, Engineering Analysis with Boundary Elements, Vol. 93, pp. 83–93 (2018).

[60] Sleijpen, G. L., Van der Vorst, H. A. and Fokkema, D. R.: BiCGstab (l) and other hybrid Bi-CG methods, Numerical Algorithms, Vol. 7, pp. 75–109 (1994).

[61] Speyer, G., Vasileska, D. and Goodnick, S.: Efficient Poisson Solver for Semiconductor Device Modeling Using the Multi-Grid Preconditioned BiCGSTAB Method, Journal of Computational Electronics, Vol. 1, pp. 359–363 (2002).

[62] Sun, J., Peterson, G. D. and Storaasli, O. O.: High-Performance Mixed-Precision Linear Solver for FPGAs, IEEE Transactions on Computers, Vol. 57, No. 12, pp. 1614–1623 (2008).

[63] Tadano, H. and Sakurai, T.: On Single Precision Preconditioners for Krylov Subspace Iterative Methods, Large-Scale Scientific Computing, pp. 721–728 (2008).

[64] 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).

[65] van der Vorst, H. A.: Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems, SIAM Journal on Scientific and Statistical Computing, Vol. 13, No. 2, pp. 631–644 (1992).

[66] van der Vorst, H. A.: Iterative Krylov methods for large linear systems, Cambridge University Press (2003).

[67] Vuik, C., van Nooyen, R. and Wesseling, P.: Parallelism in ILU-preconditioned GMRES, Parallel Computing, Vol. 24, No. 14, pp. 1927–1946 (1998).

[68] Xu, X.: OpenMP parallel implementation of stiffly stable time-stepping projection/GMRES(ILU(0)) implicit simulation of incompressible fluid flows on shared-memory, multicore architecture, Applied Mathematics and Computation, Vol. 355, pp. 238–252 (2019).

[69] Yamazaki, I., Carson, E. and Kelley, B.: Mixed Precision s-step Conjugate Gradient with Residual Replacement on GPUs, 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 886–896 (2022).

[70] Yang, L. and Brent, R.: The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, Fifth International Conference on Algorithms and Architectures for Parallel Processing, 2002. Proceedings., pp. 324–328 (2002).

[71] 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).

[72] Zhao, Y., Fukaya, T. and Iwashita, T.: Numerical Behavior of Mixed Precision Iterative Refinement Using the BiCGSTAB Method, Journal of Information Processing, Vol. 31, pp. 860–874 (2023).

[73] Zhao, Y., Fukaya, T., Zhang, L. and Iwashita, T.: Numerical Investigation into the Mixed Precision GMRES(m) Method Using FP64 and FP32, Journal of Information Processing, Vol. 30, pp. 525–537 (2022).