CCF Transactions on High Performance Computing (2023) 5:191–209
https://doi.org/10.1007/s42514-023-00141-3

REGULAR PAPER

XHYPRE: a reliable parallel numerical algorithm library for solving large-scale sparse linear equations

Chuanying Li[1] · Stef Graillat[2] · Zhe Quan[1] · Tong-Xiang Gu[4] · Hao Jiang[3] · Kenli Li[1]

Received: 30 July 2022 / Accepted: 14 March 2023 / Published online: 4 April 2023
© China Computer Federation (CCF) 2023

Abstract

With the rapid development of supercomputers, large-scale computing has become increasingly widespread in various scientific research and engineering fields. Meanwhile, the precision and efficiency of large-scale floating-point arithmetic have always been a research hotspot in high-performance computing. This paper studies the numerical method to solve large-scale sparse linear equations, in which the accumulation of rounding errors during the solution process leads to inaccurate results, and large-scale data makes the solver produce a long running time. For the above issues, we use error-free transformation technology and mixed-precision ideas to construct a reliable parallel numerical algorithm framework based on HYPRE, which solves large-scale sparse linear equations to improve accuracy and accelerate numerical calculations. Moreover, we illustrate the implementation details of our technique by implementing two cases. One is that we use error-free transformation technology to design high-precision iterative algorithms, such as GMRES, PCG, and BICGSTAB, which reduce rounding errors in the calculation process and make the result more accurate. The other is that we propose a mixed-precision iterative algorithm that utilizes low-precision formats to achieve higher computing power and reduce computing time. Experimental results demonstrate that XHYPRE has higher reliability and effectiveness . Our XHYPRE is on average 1.3x faster than HYPRE and reduces the number of iterations to 87.1% on average.

Keywords High-performance computing · Rounding errors · Error-free transformation technology · Mixed-precision

✉ Hao Jiang
haojiang@nudt.edu.cn

Chuanying Li
lichuanying@hnu.edu.cn

Stef Graillat
stef.graillat@sorbonne-universite.fr

Zhe Quan
quanzhe@hnu.edu.cn

Tong-Xiang Gu
txgu@iapcm.ac.cn

Kenli Li
lkl@hnu.edu.cn

[1] College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China

[2] Sorbonne Université, CNRS, LIP6, 10587, F-75005 Paris, France

[3] College of Computer, National University of Defense Technology, Changsha 410073, China

[4] Laboratory of Computationary Physics, Institute of Applied Physics and Computational Mathematics, Beijing 100094, China

1 Introduction

With the rapid development of high-performance computing and the vigorous popularization of supercomputers, more and more scientific computing applications in the current computing environment can no longer meet the needs of fast computing and higher precision. These applications usually exist in various fields of academia and industry, including aerospace, nuclear simulation, artificial intelligence, biomedicine, and information security (Bailey et al. 2012; Wei et al. 2022). Although the development of high-performance computing hardware devices is booming, the exploitation of corresponding software libraries is still underway for different scientific problems. For example, there will be inaccurate calculation results and slow computing speed when solving sparse linear algebraic equations in floating-point arithmetic (Muller et al. 2018).

Although double-precision arithmetic can get more accurate results in current high-performance computers, the speed of 64-bit floating-point operations (double-precision) is usually half that of 32-bit floating-point operations

(float-precision). The reason is that the lower precision can perform more operations per second on traditional processors (Baboulin et al. 2009). Therefore, efficiently improving the parallel execution efficiency of numerical calculations while improving the accuracy of the results is the focus of current research. Although some scholars have used heterogeneous parallel accelerators to accelerate numerical calculations (Tan et al. 2021; Yang et al. 2017), designing efficient and reliable parallel numerical algorithm libraries can significantly impact the current scientific computing field.

Currently, most high-performance computing platforms perform scientific calculations under the IEEE 754-2019 floating-point arithmetic standard. Previous works proved that inaccurate results could occur in numerical calculations (Mascarenhas and de Camargo 2017) (Cools et al. 2018; Du et al. 2017; Delgado Gracia 2020). Because rounding errors are unavoidable in floating-point operations (Stummel 1980), and it represents the numerical difference between the exact and approximate values calculated through floating-point operations. The cumulative effect of rounding errors can make the numerical computation results inaccurate and unreliable (Cools et al. 2018; Benz et al. 2012; Connolly et al. 2021), which leads to meaningless final calculation results. Especially solving the issue of numerical calculations based on large-scale and long-term, or solving some ill-conditioned problems of small-scale numerical calculations. Therefore, controlling the rounding error in floating-point operations has become the key issue to research. Error-Free Transformation (EFT) techniques showed reliable performance in solving this problem (Du et al. 2017; Delgado Gracia 2020; Graillat and Jézéquel 2020; Jiang et al. 2013), which compensates the rounding error of an individual operation back to the original calculated result through accumulation.

Although the calculation result's accuracy is improved, the cost will increase accordingly. The reason is that EFT will increase many calculation steps in the solution process when reducing the rounding error. Generally, some methods that incorporate the idea of mixed-precision are used to reduce the calculation time without reducing the accuracy (Kurzak et al. 2008; Abdelfattah et al. 2021; Abdulah et al. 2022). They usually use higher precision for calculation at critical parts and use lower precision in other regions in iterations to ensure the accuracy of global computing. The lower precision can perform more operations per second on traditional processors, and these methods have demonstrated reliability in maintaining high-precision calculations and reducing the overall running time, ultimately improving the peak performance (Carson and Higham 2018; Li et al. 2022; Lindquist et al. 2022). Motivated by this, we develop a reliable and efficient parallel numerical algorithm library called XHYPRE. We use EFT to reduce the rounding error during calculation to improve the accuracy of the calculation result and utilize the potential performance advantages of mixed precision to speed up and decrease the calculation time.

Taking the above factors into account, in this paper, we propose XHYPRE, which mainly uses EFT and mixed-precision ideas, constructing a reliable and efficient parallel numerical algorithm framework for large-scale sparse linear equations. It belongs to lightweight software. To further demonstrate the effectiveness of the above idea, we have conducted multiple sets of experiments on the supercomputer platform, and we have solved the ill-conditioned problems that the classic parallel algorithm library cannot solve. Experimental results demonstrate that our XHYPRE is on average 1.3x faster than HYPRE and reduces the number of iterations to 87.1% on average. To summarize, our main contributions are shown in the following four-fold:

- We propose high-precision algorithms, such as high-precision SpMV, GMRES, PCG, and BiCGSTAB, which reduce rounding errors in the calculation process and make the final results more accurate.
- We propose mixed-precision algorithms, such as mixed-precision GMRES, PCG, and BiCGSTAB, and utilize low-precision formats to achieve higher computing power, ultimately accelerating scientific computing.
- We propose a reliable parallel numerical algorithm algorithm library XHYPRE for large-scale sparse linear equations, it integrates a variety of algorithms, which show advantages in multiple performance indicators.
- Extensive experimental analyses are conducted to demonstrate that our XHYPRE can significantly improve the computational accuracy and speed in floating-point calculations, and XHYPRE can solve the ill-conditioned problems that HYPRE and PETSc cannot solve.

The remainder of this article are organized as follows. The Sect. 2 mainly discusses the background information of the relevant literature. The Sect. 3 mainly introduces rounding error and error-free transformation techniques. In Sect. 4, we describe our parallel numerical algorithm library of large-scale sparse linear equations in detail. We show the performance results of our XHYPRE in Sect. 5. Finally, we conclude the full paper and plan the subsequent research work in Sect. 6.

2 Related work

In this part, we mainly introduce relevant background knowledge and corresponding comments. The first is High Performance Preconditioners HYPRE (Sect. 2.1), the second is rounding errors and the error-free transformation techniques (Sect. 2.2), and the last is numerical methods utilizing mixed-precision (Sect. 2.3).

2.1 HYPRE

HYPRE (High Performance Preconditioners) is a parallel numerical calculation library used to solve large sparse linear algebraic equations. HYPRE has a parallel multigrid method, which is aimed at structured and unstructured grid problems and can be used for large-scale scientific simulations (Falgout et al. 2002). Some prior works (Falgout et al. 2006, 2006) has introduced HYPRE in detail. At present, HYPRE is one of the world's most popular numerical software packages, which is used to solve large-scale sparse linear equations on massively parallel computers. Some researchers have extended multi-grid solvers in HYPRE to millions of cores and even solved more than one trillion unknown problems (Baker et al. 2012). Jin and Mellor-Crummey (2002) used HYPRE semi-coarse multi-grid for SMG98 and optimized the node performance of SMG98. Reference (Engwer et al. 2017) studied the stencils with different complexity, and showed how to use them in HYPRE to enable more complex problems to perform stencil calculations. In addition, HYPRE can be used with other libraries to solve some specific problems (Schmidt et al. 2013).

Although HYPRE is very powerful, handling specific problems still falls short of expectations. Therefore, many researchers began to further improve HYPRE. Gershman and Strichman (2005) proposed an improved Hyper preprocessing algorithm. When HYPRE is combined with modern SAT solvers to solve a specified type of problem, Gershman's algorithm generates less information than HYPRE but is more efficient. It can spend less total running time, including the SAT solvers, especially in large-scale experiments. For the symmetric eigenvalue problem, Lashuk et al. developed the LOBPCG code for the HYPRE and PETSc software packages (Lashuk et al. 2007). The combination of HYPRE preconditioners and PTESc can facilitate a quick comparison of algorithms (Zhang et al. 2008). For the performance problem of the parallel preconditioned algorithm, Knyazev et al. used HYPRE's algebraic multigrid and preconditioners to solve it and later developed the software package Xolvers (BLOPEX), which can be used as an independent serial library or built into HYPRE (Knyazev et al. 2007). The same preprocessor can be effectively used for symmetric eigenvalue problems by BLOPEX. Sahasrabudhe et al. (2021) optimized the HYPRE solver for the core and GPU architectures. They combined OpenMP and HYPRE to reduce overheads while increasing execution speed (Sahasrabudhe and Berzins 2020). However, most of these studies are related to solvers, and they are only partially optimized. They did not take into account the accuracy of the calculations, nor the rounding errors that may occur during the calculations.

2.2 Rounding errors and error-free transformation techniques

In processing floating-point operations, rounding errors will inevitably occur, and they nearly appear in all elementary operations, significantly when solving ill-conditioned problems. After the rounding errors continue to accumulate, the calculation result will become inaccurate or even completely incorrect (Mascarenhas and de Camargo 2017; Cools et al. 2018; de Camargo 2020). Menon et al. proposed ADAPT (Menon et al. 2018), which uses algorithmic differentiation for mixed precision analysis for HPC workloads, and estimates rounding errors, then accurately estimates the output error. Connolly et al. (2021) analyzed stochastic characteristics and explained the process of rounding errors well. Moreover, Higham and Mary (2019) proposed a novel analysis method of probabilistic rounding error, which has been applied to a variety of algorithms. Ozaki et al. (2021) designed a method to verify the numerical computation of a large-scale linear system and obtained the error range of the numerical results. Nevertheless, they are limited to verifying the numerical calculation of linear systems, and they did not use an effective method to solve the problem of inaccurate calculations caused by rounding errors.

Therefore, how to control the rounding error in floating-point arithmetic has become a research focus. The compensated algorithm based on EFT shows reliable performance in solving rounding errors (Delgado Gracia 2020; Hermes 2019). For the tensor product functions, Delgado Gracia (2020) designed the compensation algorithm of de Casteljau, which is very effective in ill-conditioned situations. By using EFT techniques, Graillat and Jézéquel (2020) proposed the new approach that used compensated algorithms to obtain tight interval inclusions, then used to study the compensated algorithms. Du et al. (2017) proposed a compensated quotient-difference (Compqd) algorithm, which combines traditional QD algorithm and EFT to make the numerical results more stable. Jiang et al. (2013) and Graillat et al. (2018) evaluated and verified the compensated algorithm using EFT. Jiang et al. evaluated the compensated algorithm for the k-th derivative of a polynomial, Graillat et al. used stochastic arithmetic to verify the result of the compensated algorithm, and the experiment results of the two proved the accuracy of the compensated algorithm based on EFT. In our XHYPRE, we use the EFT design algorithm to make XHYPRE's results more accurate and stable.

2.3 Numerical methods utilizing mixed-precision

Almost all numerical calculations are performed utilizing floating-point data types. The accuracy of these data types must reduce overall rounding errors while emphasizing

performance improvements. Generally, mixed-precision can achieve this excellent effect. Mixed-precision uses higher precision calculations for key operations when working low-precision, accelerating computations while maintaining high-precision output. Carson and Higham (2018) combined the three precisions for iterative refinement. Li et al. (2023, 2021) utilized mixed precision to design a calculation framework with adjustable precision to solve singular values, so as to improve the calculation speed and enhance the overall performance. Sorna et al. (2018) designed the mixed-precision Fast Fourier Transform algorithm, which committed to accelerating calculations while maintaining their calculation accuracy.

In addition, linear solvers can also apply mixed-precision ideas to improve performance (Sun et al. 2008; McCormick et al. 2021). Blanchard et al. (2020) designed a mixed-precision block FMA and applied it to matrix multiplication and LU factorization. Haidar et al. (2020) designed mixed-precision LU factorization and Generalized Minimal Residual Method (GMRES) using low precision on NVIDIA GPUs to speed up the equation solving while maintaining numerical stability. Lindquist et al. (2022) designed a mixed-precision GMRES, which achieved the same convergence and increased speed compared with double-precision GMRES. The Multiple Relatively Robust Representations algorithm using mixed-precision methods can also improve the accuracy and reduce the calculation time (Petschow et al. 2014). Based on the current advantages of mixed-precision, we also utilize the mixed-precision idea to design a parallel numerical algorithm library to solve large-scale sparse linear equations efficiently.

3 Preliminary

At present, the floating-point arithmetic strategy is currently the most widely used approximate real number algorithm. The computer uses IEEE Standard for Binary Floating-Point Arithmetic for scientific calculations, and rounding errors will inevitably occur in the floating-point operations. Meanwhile, during the execution of large-scale calculations, even a minimal error after a considerable accumulation will make the result inaccurate or completely incorrect. Especially in a parallel numerical algorithm library of large-scale sparse linear equations, the algorithm is changed by fusing arithmetic strategies and mixed_precision ideas to control the accumulation of rounding errors, thereby improving the accuracy of the numerical algorithm. In this section, we first introduce rounding errors in floating-point arithmetic (Sect. 3.1), and then introduce error-free transformation techniques that effectively control rounding errors (Sect. 3.2)).

3.1 Rounding errors

The difference between the exact value and the approximate value obtained by floating-point operations is the rounding error, which causes the instability of the numerical result. In the IEEE-754 floating-point standard, the unit roundoff for binary32 is approximately equal to 10^{-8}, and the unit roundoff for binary64 is approximately equal to 10^{-16}.

Within the scope of floating-point numbers, a standard model of floating-point operation is denoted as follow:

$$x\ op\ y = fl(x \circ y) = (x \circ y)(1+\varepsilon_1) = (x \circ y)/(1+\varepsilon_2), \tag{1}$$

where $x \in \mathbb{R}$, $y \in \mathbb{R}$, and op indicates the floating-point addition, subtraction, multiplication, and division operations inside the computer, fl represents floating-point operation, and $\circ \in \{+,-,\times,\div\}, |\varepsilon_1| \le \frac{u}{1+u}, |\varepsilon_2| \le \frac{u}{1+u}$. Here u is the working precision of the machine used in basic arithmetic operations, which is the unit roundoff. Some documents regard u as machine epsilon, abbreviated as **eps**. For rounding to nearest mode, there is $u = \mathbf{eps/2}$. For the other three rounding modes, $eps = \mathbf{u}$. In IEEE standard 754, double-precision rounding to nearest mode, $eps = 2^{-52}$, and $u = 2^{-53}$.

Without the situation of underflow, the error bounds of the basic floating-point operations model are obtained by the following equation:

$$\begin{aligned} &|x \circ y - fl(x \circ y)| \le \mathbf{u}|x \circ y|, \qquad and \\ &|x \circ y - fl(x \circ y)| \le \mathbf{u}|fl(x \circ y)|. \end{aligned} \tag{2}$$

This is the basic process of rounding errors in calculations due to the limited word length of the computer.

3.2 Error-free transformation techniques

Oishi, Ogita, and Rump formally proposed the concept of Error-Free Transformation (EFT) in 2005 (Ogita et al. 2005). It becomes an important tool for the later design of compensated algorithms. An increasing number of compensation algorithms have thus flourished. For example, Graillat and Ménissier-Morain (2008) proposed a compensated Horner scheme for accurately calculating the polynomial evaluation.

Suppose $\circ \in \{+,-,\times\}$, two floating-point numbers $(x, y) \in \mathbb{F}$, and $a = fl(x \circ y) \in \mathbb{F}$, $b \in \mathbb{F}$. When there is no situation of overflow and underflow, the rounding mode is

set to rounding to nearest, and then the equation is denoted as follows:

$$(x \circ y) = a + b, \tag{3}$$

where a represents the best floating-point approximation of the calculation result, and b represents the exact rounding error. The focus of EFT techniques is to calculate the value of b accurately. The process of satisfying the above expression is the EFT of a pair of floating-point number addition, subtraction and multiplication operations.

3.2.1 EFT of the sum of two floating-point numbers

Suppose two floating-point numbers $(x, y) \in \mathbb{F}$, and sum of floating-point numbers is $a = fl(x \circ y) \in \mathbb{F}$, rounding error is $b \in \mathbb{F}$. The TwoSum algorithm requires six floating-point operations, which was proposed by Knuth (2014). The computational details are presented in Algorithm 1. The TwoSum algorithm does not require a prior condition, and it still holds when underflow occurs. And it is the best algorithm known so far.

3.2.2 EFT of the product of two floating-point numbers

We use the TwoProd algorithm (Dekker 1971), but the Fused-Multiply-and-Add (FMA) (Muller et al. 2010) unit shows a smaller area and higher accuracy than a separate multiplier and adder, which can execute floating-point multiplication and addition operations simultaneously. Specifically, the FMA is a floating-point multiply-add operation that performs $n + p \times q$ in one step, where $n, p, q \in \mathbb{F}$. It is a common arithmetic operation that only performs rounding once. The non-FMA operation is to first calculate the result of $p \times q$, round the result to M significant bits, then add the result to n, and finally round the added result to M significant bits. The FMA calculates the value of $n + p \times q$ with full precision, and then rounds the final result to M significant bits. It can be seen that FMA has the advantages of fast calculation speed and high calculation precision compared with the traditional floating-point adder and floating-point multiplier to perform floating-point multiplication and addition operations. Therefore, we changed the TwoProd algorithm to the TwoProdFMA algorithm and applied it to our parallel numerical algorithm library to reduce the impact of rounding errors. The TwoProdFMA algorithm only costs 2 flops, as shown in Algorithm 2.

Algorithm 1 Error-free transformation for the summation of two floating-point numbers

Require: $x \in \mathbb{F}, y \in \mathbb{F}$
Ensure: Sum of floating-point numbers: a, Rounding error: b
1: function : $[a, b] = \text{TwoSum}(x, y)$
2: $a = x \oplus y$
3: $z = a \ominus x$
4: $b = (x \ominus (a \ominus z)) \oplus (y \ominus z)$

Algorithm 2 TwoProdFMA Algorithm

Require: $x \in \mathbb{F}, y \in \mathbb{F}$
Ensure: Product of floating-point numbers: a, Rounding error: b
1: function $[a, b] = \text{TwoProdFMA}(x, y)$
2: $a = x \otimes y$
3: $b = \text{FMA}(x, y, -a)$

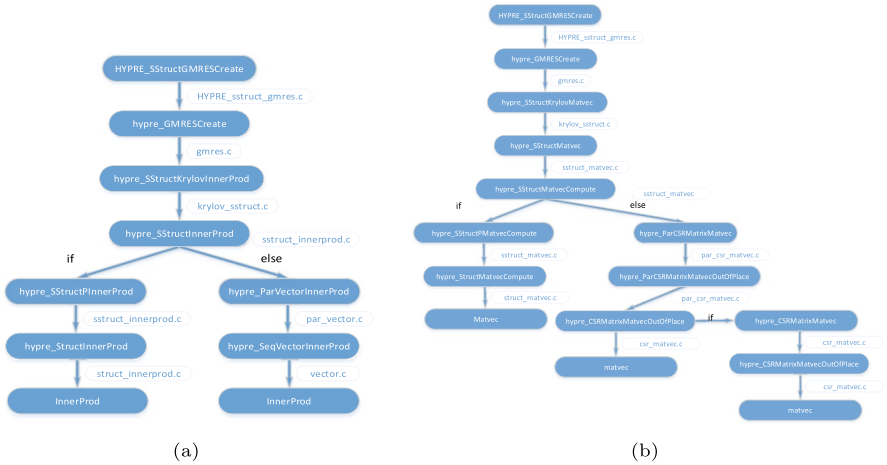

Fig. 1 The execution flow chart of the hotspot function

4 A reliable and efficient numerical algorithm library

XHYPRE is a reliable and efficient parallel numerical algorithm framework designed and implemented based on HYPRE [54]. The central idea of XHYPRE is EFT technology and mixed-precision strategy. In what follows, we first introduce the general design of our proposed parallel numerical algorithm library (Sect. 4.1). Then give an example of how to use EFT technology to improve the accuracy of the calculation in Sect. 4.2, and Sect. 4.3 shows an example of how to utilize the idea of mixed precision to speed up.

4.1 Overview

In the following part, we introduce a reliable and efficient iterative refinement numerical algorithm framework. In particular, the original method is not feasible when solving ill-conditioned problems.

The core strategy of this framework is to refine the algorithm's solving process, use EFT technology to improve the calculation accuracy, and utilize the idea of mixed precision to accelerate the calculation process. Therefore, we perform performance analysis of the application and analyze the implementation process of many algorithms in HYPRE.

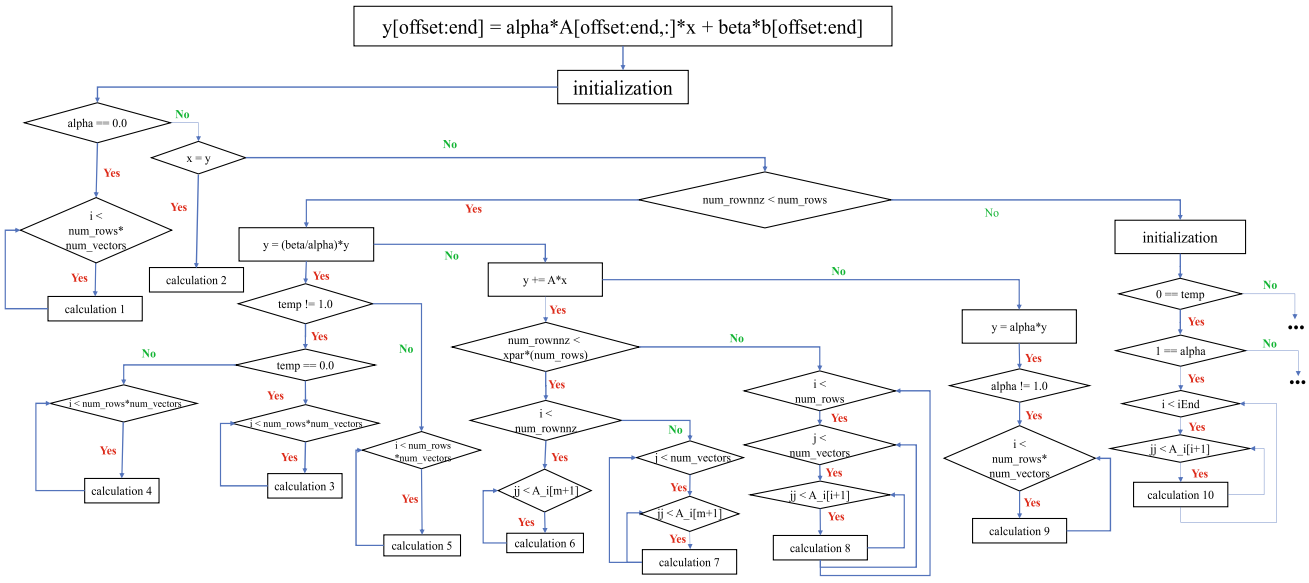

Fig. 2 The execution flow of SpMV

We select one of the hotspot functions as an example to show the execution flow of the function, as shown in Fig. 1. This hotspot function uses the GMRES algorithm of the Krylov subspace, adopts the SStruct interface, and has two different implementations according to the difference of the Data Layout. It can be seen from the Fig. 1 that there are two kinds of Data Layout, structured and CSR. In our analysis of the iterative refinement of the algorithm, we find that the algorithm's dot product and sparse matrix–vector multiplication (SpMV) during loop iteration are computationally intensive and require high computational accuracy. Meanwhile, we can see from Fig. 1 that the dot product and SpMV are the bottom-level computation functions of the hotspot function. And according to different judgment conditions, their implementation methods are also other. We take one of the SpMV as an example, and its execution flow is shown in Fig. 2. Note that calculation $*$ represents different calculation methods according to different judgment conditions, where $*\in [1:10]$.

When we change the dot product and SpMV and do not make changes to other parts, we find that they significantly impact the algorithm's overall number of iterations and running time after experimental verification. Especially in floating-point arithmetic, due to the increase of the data scale, which dramatically increases the calculation time, the accumulated rounding error will cause the calculation result to seriously deviate from expectations and fail to achieve the best convergence state. Therefore, we improve the overall performance by decreasing the rounding error of the dot product and SpMV.

Firstly, we analyze the cumulative effect of rounding errors. We use EFT technology to compensate for the rounding error back to the previous computation result for dot product and SpMV, which improves the calculation accuracy and reduces the number of iterations. Although we can directly utilize a higher precision data format to calculate dot product and SpMV, it is not suitable for all machines. So, we choose to use EFT technology to make our XHYPRE universality while ensuring calculation accuracy. Then we adopt the mixed-precision idea, which is running the algorithm at reduced precision while maintaining high precision for critical operations, as far as possible to utilize the lower precision data format to obtain higher performance. Specifically, the dot product and SpMV are used with full-precision, while other parts are used with reduced-precision. The reduced-precision part reduces the runtime, and the full-precision part ensures the accuracy of the final result. Finally, we construct a reliable and efficient parallel numerical algorithm library named XHYPRE. The XHYPRE software library is shown in Fig. 3.

In XHYPRE, we design new algorithms to provide new interfaces for the solver, which can solve large sparse linear equations and ill-conditioned problems on a massively parallel machine. It includes parallel implementations of several types of typical iteration methods (such as Krylov subspace iteration methods). We take Krylov subspace iteration

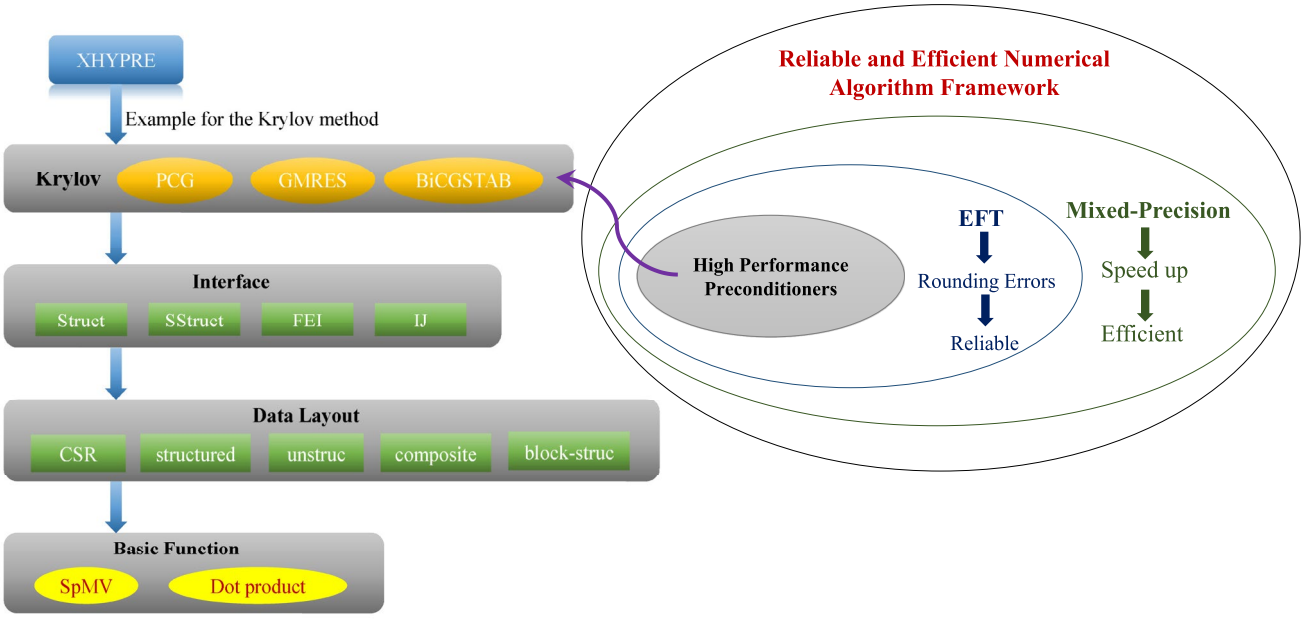

Fig. 3 The Framework of XHYPRE

methods as an example to show the calculation process of XHYPRE in Fig. 3. We first select the algorithm that can solve the problem, then transfer the data through the function interface, and finally calculate the final result.

Our XHYPRE software library relies on HYPRE library, as shown in Fig. 3. Running the script x_hypre.sh in our library can transform the HYPRE library into the XHYPRE library. Our script manages all the implementation details, including the method's implementation to the function's declaration in the header file, and the complex calling relationship of the function in the makefile. Users can implement complex code transformations by simply running x_hypre.sh, which reduces the difficulty for users to use.

Our XHYPRE software has the following advantages. First, our XHYPRE belongs to lightweight software, which is easy to download and has high flexibility. For example, the programmer can still use any preconditioners and functional interfaces after performing the transformation or choose to use our method. Second, this method is less complicated for any user, the operation is more straightforward, and the choice of functions is more diverse. Giving users more freedom and space to utilize XHYPRE in various scenarios. Moreover, we support reproducible science. XHYPRE is available as a free open-source numerical algorithm library on GitHub. It contains two versions, including the version integrated with TwoProd algorithm (https://github.com/compilerOpt/-XHYPRE−1.0.0) and the version integrated with Fused-Multiply-and-Add (FMA) (https://github.com/compilerOpt/-XHYPRE−2.0.0).

The following numerical test introduces the performance of XHYPRE from the following two aspects (Sect. 5). One is that we use EFT technology to reduce the accumulation of rounding errors and design reliable numerical algorithms to make the output results more accurate and stable, it can be called high-precision XHYPRE. The other is that we leverage mixed-precision to speed up calculations and improve overall performance, it can be called mixed-precision XHYPRE. Furthermore, we take the GMRES and Preconditioned Conjugate Gradient (PCG) as an example, and the two aspects of XHYPRE are introduced in detail through examples in Sect. 4.2 and Sect. 4.3. Although our XHYPRE contains multiple algorithms, such as the Biconjugate gradient stabilized method (BiCGSTAB), we will not introduce them one by one due to using the same implementation strategy.

4.2 Implementation of high-precision algorithm

We analyze the application's code and choose the code that can improve the performance of the software for optimization. The above analysis shows that the dot product and SpMV are the hotspot functions. We analyze the hotspots' code and find that the rounding errors caused by vector multiplication and addition are discarded in the original solution process of SpMV, which makes the calculation have deviation, ultimately leading to inaccurate results.

Algorithm 3 SpMV in XHYPRE (XSpMV)

Require: $nrows$, A_value, A_index, $A\_pointer$, x
Ensure: Vector: b

1: **for** $i = 0$ **to** $nrows - 1$ **do**
2: Initialize $th = 0, temp = 0$
3: **for** $j = A\_pointer[i]$ **to** $j = A\_pointer[i + 1] - 1$ **do**
4: $[sh, sl] = \text{TwoProdFMA}(A\_value[j], x[A\_index[j]])$
5: $[th, tl] = \text{TwoSum}(th, sh)$
6: $temp = temp + (sl + tl)$
7: **end for**
8: $b_i = temp + th$
9: **end for**

In order to improve the accuracy of the calculation, we use the Dot2 algorithm in reference (Ogita et al. 2005) for computing the dot product. Based on this, we propose a high-precision SpMV algorithm. The original solution is combined with EFT techniques to form a high-precision SpMV named XSpMV. The high-precision SpMV's execution process is shown in Algorithm 3. The *nrows* is the number of rows in the matrix, vector *A_value* records the values of each non-zero element, vector *A_index* records the column indices of the non-zero elements, vector *A_pointer* records the index of the first non-zero element of each row in vector *A_index* and vector *A_value*.

First, we use the TwoProdFMA algorithm to obtain the product of each element of the matrix *A* and the vector *x*. Then, we utilize the TwoSum algorithm to sum the ordinary floating-point results, and add up the errors *res* they initially discarded. Finally, the floating-point result *th* obtained is added to each error *res*. Similarly, each row of matrix *A* is multiplied by vector *x* in this way. We changed the code for each part, including the high-precision calculation of the dot product. We use the typedef keyword to create the data type of *dd_real*, which includes high-order and low-order parts of the data, equivalent to implementing 128-bit calculations. But our input and output are both double data, and only high-precision calculations are implemented internally. We tested the XSpMV, and the test results proved that the modification's effect was obvious.

We mainly combine some algorithms with EFT techniques to turn them into higher precision algorithms, ultimately improving the accuracy of calculations. From the analysis in Sect. 4.1, we can see that the essential parts that affect the performance of these algorithms are dot product and SpMV. We take the GMRES and PCG algorithms as examples for a brief description. The high-precision GMRES is shown in Algorithm 4. Its input and output keep unchanged. However, the dot product and SpMV use EFT technology to become a high-precision solution, reducing rounding errors and making the calculation more accurate. Therefore, our HGMRES algorithm can solve the ill-conditioned problem that HYPRE and PETSc cannot solve (Sect. 5.1). We also show the high-precision PCG algorithm in Algorithm 5. The high-precision PCG algorithm is more accurate in each layer iterative calculation, because the EFT technology compensates the accumulated rounding error back.

Algorithm 4 High-precision Generalized Minimal Residual method (HGMRES)

Require: Matrix A, vector b
Ensure: Approximate solution of linear system $Ax = b$
1: Compute $r_0 = b - \text{XSpMV}(A, x_0)$, $\beta = \sqrt{\text{Dot2}(r_0, r_0)}$, and $v_1 = r_0/\beta$
2: **for** $t = 1$ **to** T **do**
3: Compute $w_t = \text{XSpMV}(A, v_t)$
4: **for** $m = 1$ **to** t **do**
5: $h_{mt} = \text{Dot2}(w_t, v_t)$
6: $w_t = \text{Dot2}(w_t, h_{mt}v_t)$
7: **end for**
8: $h_{t+1,t} = \sqrt{\text{Dot2}(w_t, w_t)}$. If $h_{t+1,t} == 0$ set $T = t$ and go to 11
9: $v_{t+1} = w_t/h_{t+1,t}$
10: **end for**
11: Define the $(T+1) \times T$, Hessenberg matrix $\overline{H}_T = \{h_{mt}\}_{1 \le m \le m+1, 1 \le t \le T}$
12: Compute y_T the minimizer of $\left\|\beta e_1 - \overline{H}_T y\right\|_2$ and $x_T = x_0 + V_T y_T$

Algorithm 5 High-precision Preconditioned Conjugate Gradient method (HPCG)

Require: Matrix A, vector b
Ensure: Approximate solution of linear system $Ax = b$
1: Compute $r_0 = b - \text{XSpMV}(A,x_0)$, $z_0 = M^{-1}r_0$, and $p_0 = z_0$
2: **for** t =0,1,..., until convergence **do**
3: $\alpha_t = \text{Dot2}(r_t, z_t)/\text{Dot2}(Ap_t, p_t)$
4: $x_{t+1} = x_t + \alpha_t p_t$
5: $r_{t+1} = r_t - \alpha_t \text{ XSpMV}(A, p_t)$
6: $z_{t+1} = M^{-1}r_{t+1}$
7: $\beta_t = \text{Dot2}(r_{t+1}, z_{t+1})/\text{ Dot2}(r_t, z_t)$
8: $p_{t+1} = z_{t+1} + \beta_t p_t$
9: **end for**

4.3 Implementation of mixed-precision algorithm

As the computation scale of high-performance computing increases, it becomes increasingly essential to decrease runtime and memory bandwidth without reducing the overall precision. Therefore, developing algorithms that utilize mixed-precision algorithms to provide higher performance can significantly impact HPC. This is equivalent to approximate computing that balances program accuracy and speed, and ultimately improves the peak performance.

Algorithm 6 Mixed-precision Generalized Minimal Residual method (MGMRES)

Require: Matrix A, vector b;
Ensure: Approximate solution of linear system $Ax = b$;
1: Compute $r_0 = b - Ax_0$ (u_{high})
2: $\beta = \|r_0\|_2$ (u_{high})
3: $v_1 = r_0/\beta$; (u_{low})
4: **for** $t = 1$ **to** T **do**
5: Compute $w_t = Av_t$; (u_{high})
6: **for** $m = 1$ **to** t **do**
7: $h_{mt} = (w_t, v_t)$; (u_{high})
8: $w_t = (w_t, h_{mt}v_t)$; (u_{high})
9: **end for**
10: $h_{t+1,t} = \|w_t\|_2$ (u_{high}) (u_{low})
11: If $h_{t+1,t} == 0$ set $T = t$ and go to 14
12: $v_{t+1} = w_t/h_{t+1,t}$; (u_{low})
13: **end for**
14: Define the $(T + 1) \times T$, Hessenberg matrix $\overline{H}_T = \{h_{mt}\}_{1 \le m \le m+1, 1 \le t \le T}$. (u_{low})
15: Compute y_T the minimizer of $\left\|\beta e_1 - \overline{H}_T y\right\|_2$ and $x_T = x_0 + V_T y_T$ (u_{low})

Table 1 Specific information of the matrix

Name	Order	Non-zero element	Application area	Convergence
Solverchallenge21_01	2,097,152	14,581,760	Laser fusion	<1e-10
Solverchallenge21_02	6,291,456	52,133,888	Laser fusion	<1e-10
Solverchallenge21_03	83,073	2,826,927	Engineering mechanics	<1e-8
Solverchallenge21_04	2,081,541	71,033,481	Engineering mechanics	<1e-8
Solverchallenge21_05	225,800	3,591,872	Electronics system	<1e-8

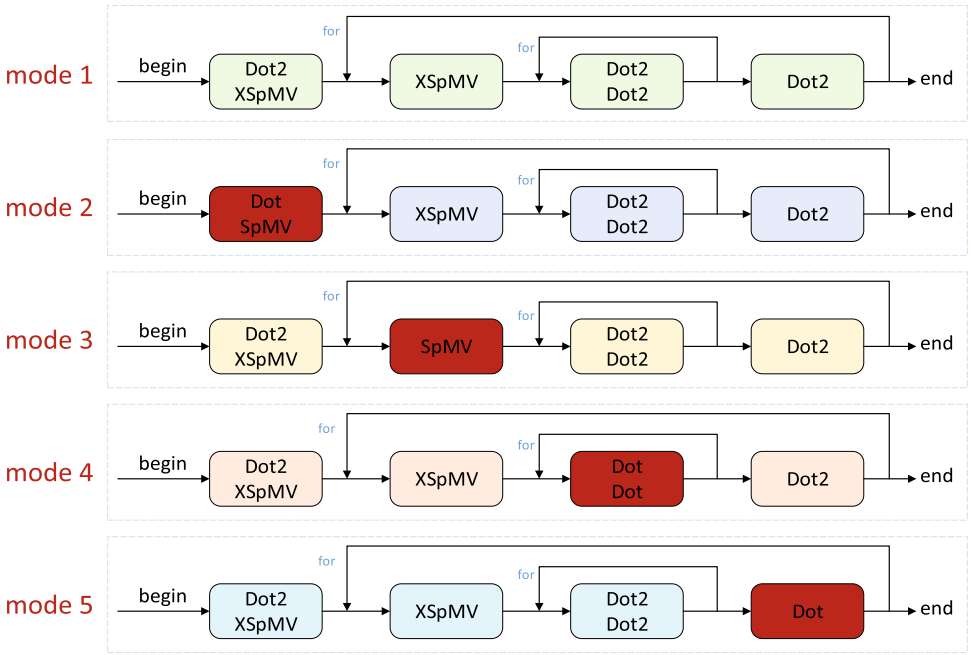

Fig. 4 Flow chart of five modes of the GMRES algorithm

Generally, a single-precision algorithm is used to calculate the start and end, and a double-precision algorithm is used to calculate the refinement process. The single-precision part reduces the runtime and saves memory bandwidth, and the double-precision part ensures the accuracy of the final result. Based on the above research, we focus on a specific mixed-precision strategy in our XHYPRE, which has achieved successful results in both accuracy and performance.

In XHYPRE, the accuracy of these data types must reduce overall rounding errors while emphasizing performance improvements. We use the GMRES and PCG algorithm as an example to illustrate the mixed-precision idea. Specifically, the dot product and SpMV are used with full-precision, while other parts are used with reduced-precision. In this paper, u_{low} stands for use float-precision computation, u_{high} stands for use double-precision computation. Mixed precision GMRES as shown in Algorithm 6. We utilize low-precision data formats as much as possible to achieve high computing accuracy.

Algorithm 7 Mixed-precision Preconditioned Conjugate Gradient method (MPCG)

Require: Matrix A, vector b;
Ensure: Approximate solution of linear system $Ax = b$;
1: Compute $r_0 = b - Ax_0$ (u_{high})
2: $z_0 = M^{-1}r_0$, and $p_0 = z_0$; (u_{low})
3: **for** t =0,1,..., until convergence **do**
4: $\alpha_t = (r_t, z_t)/(Ap_t, p_t)$ (u_{high})
5: $x_{t+1} = x_t + \alpha_t p_t$ (u_{low})
6: $r_{t+1} = r_t - \alpha_t Ap_t$ (u_{high}) (u_{low})
7: $z_{t+1} = M^{-1}r_{t+1}$ (u_{low})
8: $\beta_t = (r_{t+1}, z_{t+1})/(r_t, z_t)$ (u_{high})
9: $p_{t+1} = z_{t+1} + \beta_t p_t$ (u_{low})
10: **end for**

Table 2 Convergence results of XHYPRE (number of iterations)

Name	HYPRE	PETSc	XHYPRE
Solverchallenge21_01	Not convergent	Not convergent	4431
Solverchallenge21_02	17	69	17
Solverchallenge21_03	Not convergent	Not convergent	704
Solverchallenge21_04	3	3	3
Solverchallenge21_05	1	1	1

At the same time, the mixed-precision PCG algorithm is shown in Algorithm 7. We iteratively refine the PCG algorithm, double precision calculation key parts, other parts utilize float precision to calculate, and then achieve better performance. Finally, we realized a set of efficient floating-point numerical algorithms to speed up. And we have verified our ideas by conducting experiments on the supercomputing platform.

5 Experimental evaluation

This section comprehensively evaluates the performance of our XHYPRE. We conducted detailed experiments on the Tianhe-1 supercomputer. Tianhe-1 has 2048 nodes. The computing node has two Intel Xeon Westmere EP 2.50GHz CPUs. Each CPU has 6 cores, 48GB memory, single computing node CPU peak performance 140.64 GFlops. The data used for calculation are all floating-point numbers. All numerical experiments in this paper are carried out under

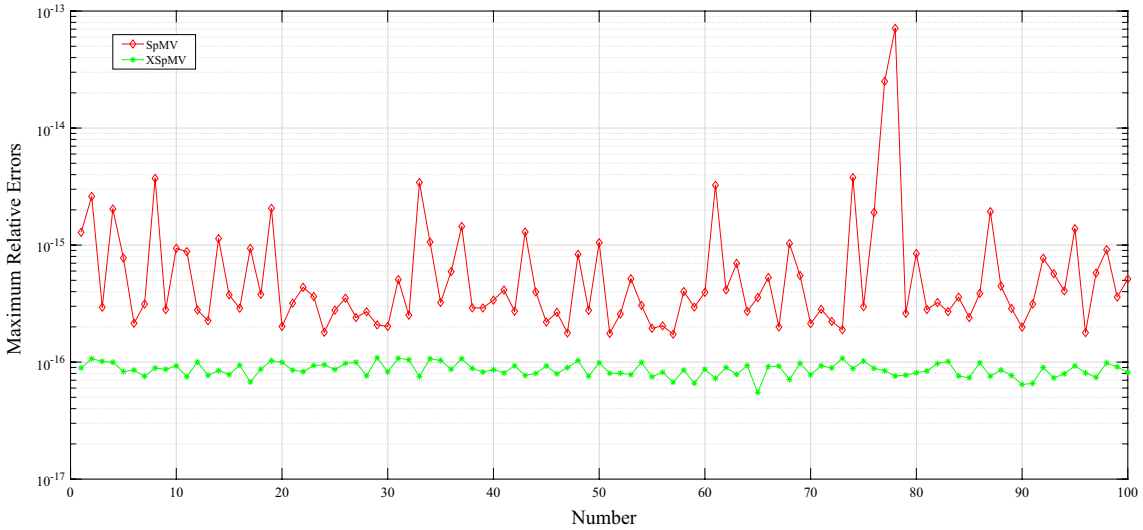

Fig. 5 Maximum relative errors on the vector *b* using the SpMV and XSpMV algorithms, the number of matrices is set to 100

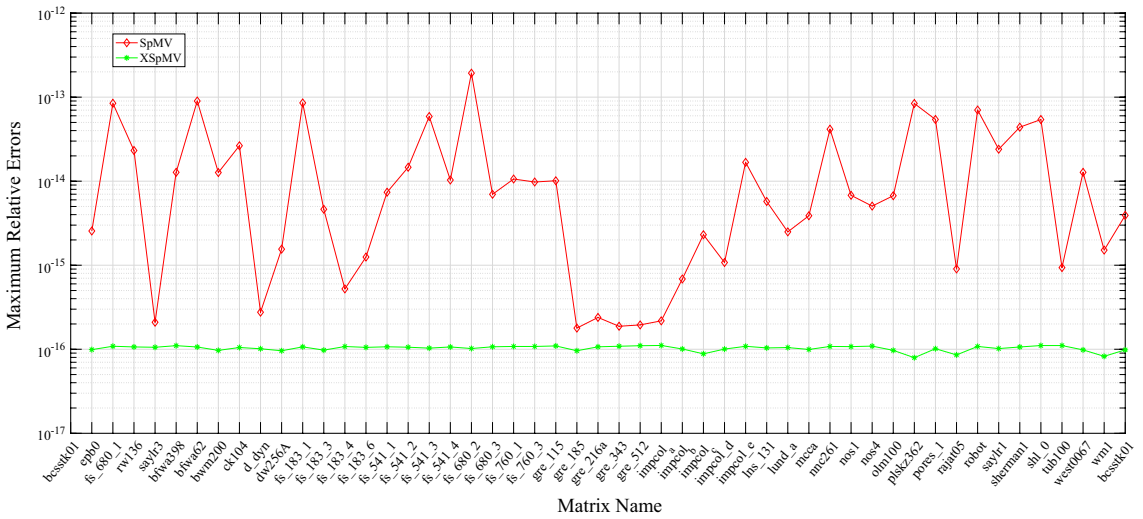

Fig. 6 Maximum relative errors on the vector *b* using the SpMV and XSpMV algorithms

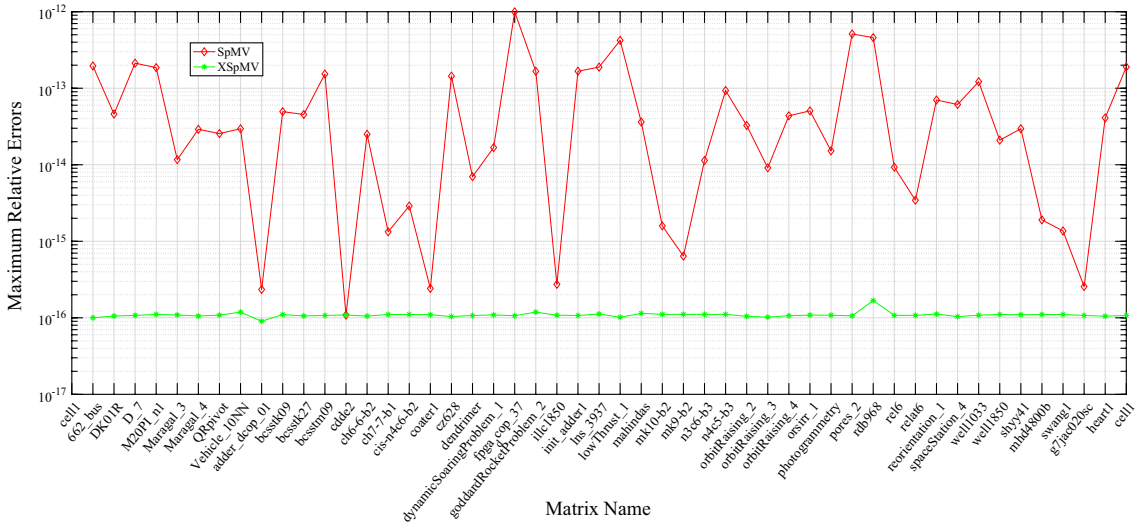

Fig. 7 Maximum relative errors on the vector *b* using the SpMV and XSpMV algorithms

the IEEE-754 standard. The software configuration includes GCC v8.3.0 and MPICH v3.3.2.

5.1 Convergence verification XHYPRE

We select five ill-conditioned matrices [55] for testing to prove that the convergence of XHYPRE is better than that of HYPRE. Where the solverchallenge21_01 and the solverchallenge21_02 are derived from the three-dimensional photon equation of radiation fluid mechanics and are a structural grid. The solverchallenge21_03 matrix is derived from the linear elastic equation of the contact mechanics of the centrifuge device and is a first-order nodal finite element. The solverchallenge21_04 matrix is derived from the linear elastic equation of contact mechanics simulated of the aero-engine; its discretization format is first-order nodal finite element. And the solverchallenge21_05 comes from filter electromagnetic field simulation time-harmonic Maxwell equations; Its discretization format is a first-order Nedelec finite element. Table 1 shows the detailed information of the matrix.

PETSc (Portable, ExtensibleToolkit for Scientific Computation) [56] provides almost all efficient solvers for solving linear equations, serial and parallel solutions, and direct and iterative solutions. For large-scale linear equations, PETSc provides many mature and effective iterative methods based on the Krylov subspace method and various preconditioners, as well as interfaces to other general programs and user programs.

We test PETSc, HYPRE, and XHYPRE on the Tianhe-1 supercomputer. Table 2 shows the test results. For the solverchallenge21_01 matrix, HYPRE and PETSc cannot converge, while our XHYPRE converges after 4431 iterations without preconditions. For the solverchallenge21_03 matrix, HYPRE and PETSc cannot converge, while our XHYPRE can converge after 704 iterations without preconditions. For the three matrices of solverchallenge21_02, solverchallenge21_04, and solverchallenge21_05, PETSc, HYPRE, and XHYPRE can all solve them successfully. It should be noted that for the two matrices of solverchallenge21_01 and solverchallenge21_03, only XHYPRE can meet the accuracy requirements and converge at the same time. So, the XHYPRE can solve the above two linear systems, while the HYPRE and PETSc can not.

In order to fully illustrate the convergence of our algorithm, we take XGMRES as an example. We change XGMRES in 4 different modes and conducted experiments on them, respectively. The flow chart of the GMRES algorithm in 5 different modes is shown in Fig. 4. Note: This flowchart only involves dot and SpMV. As shown in Algorithm 7, mode 1 is the flow chart of our XGMRES algorithm, mode 2 – mode 5 is to modify the part process of the XGMRES algorithm and restore it to the algorithm used by GMRES. It can be seen from Table 2 that our XHYPRE using the XGMRES algorithm can solve two linear systems (mode 1), but the original HYPRE using the GMRES algorithm cannot. Therefore, we conduct experiments on the GMRES algorithm in four modes.

After experimental verification, it can be known that the XGMRES algorithm can not be convergent no matter which module is changed. Only mode 1 can be convergent, which can prove the convergence and effectiveness of XGMRES. So, our XHYPRE can solve ill-conditioned problems that HYPRE and PETSc cannot solve.

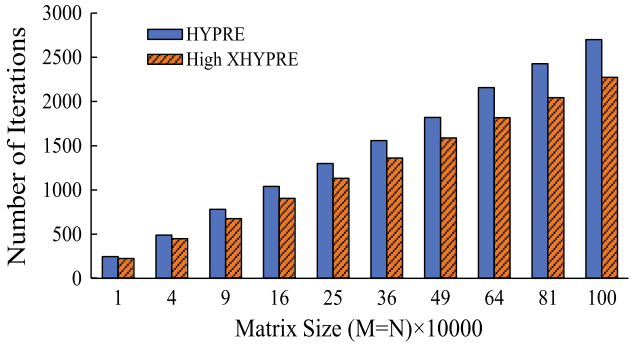

Fig. 8 The number of iterations of HYPRE and High XHYPRE

Table 3 The computation time of HYPRE and High XHYPRE on the CPU

Matrix size	HYPRE (ms)	High XHYPRE (ms)
1×10^4	38.78	51.19
4×10^4	175.05	384.46
9×10^4	748.43	1275.98
16×10^4	1449.56	2131.64
25×10^4	3798.33	5787.48
36×10^4	7138.96	12136.52
49×10^4	11073.22	18472.98
64×10^4	20261.81	32381.51
81×10^4	28919.92	43746.63
100×10^4	42088.86	55041.13

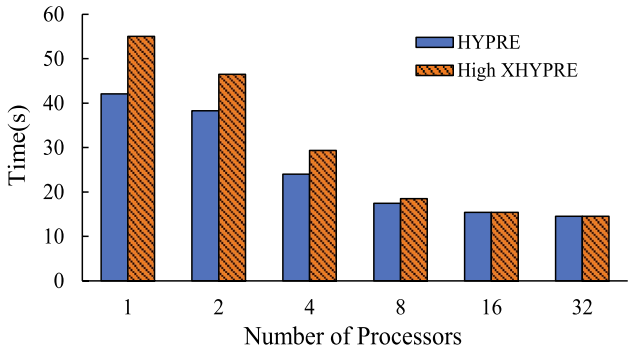

Fig. 9 The computation time of multi-processor HYPRE and High XHYPRE on the CPU, matrix size is $M=N=1\times10^6$

5.2 Precision analysis of XSpMV

This section evaluates the performance of the XSpMV algorithm by comparing the relative error and running time of ordinary SpMV (SpMV) and high-precision SpMV (XSpMV) at different scales. In addition, we use Symbolic Toolbox in Matlab to calculate the 'exact' results as the criterion.

Table 4 The number of iterations of FP32 HYPRE, Mixed XHYPRE and HYPRE

Matrix size	FP32 HYPRE	Mixed XHYPRE	HYPRE
16×10^2	76	63	63
25×10^2	103	79	79
36×10^2	124	95	95
49×10^2	144	111	111
64×10^2	173	127	127
81×10^2	193	143	143
100×10^2	243	159	159

Firstly, we test 100 sparse matrix–vector multiplication of different sizes to demonstrate the accuracy and stability of the XSpMV better. We select 100 matrices and vectors of different sizes. The matrix size is $m * n$ (m and n are integers randomly obtained by *rand* in the interval (20, 2000)), and the vector size is n. We use sparse matrixes that obey a uniform distribution. In this section, we set the distribution density of non-zero elements as 0.6. The values of matrices and vectors are generated by the GenDot algorithm (Ogita et al. 2005). In Fig. 5, we show the maximum relative error of all elements in each result vector b_n obtained using the SpMV and XSpMV algorithm, respectively. The number in the figure represents the *Nth* sparse matrix–vector multiplication, the vertical axis is the maximum relative error of the result vector $b_m^{(N)}$. The calculation formula comes from literature (Du et al. 2017), that is

$$\mathrm{MRE} = \max_m \max_N \frac{\left|r_m^{(N)} - b_m^{(N)}\right|}{\left|r_m^{(N)}\right|}, \tag{4}$$

where $N = 1 : 100$. *MRE* stands for the maximum relative error of the result vector $b_m^{(N)}$. $r_m^{(N)}$ represents the 'exact' result calculated using the Symbolic Toolbox in Matlab, and $b_m^{(N)}$ represents the calculation result of the algorithm. One can see from Fig. 5 that the relative error computed by the SpMV fluctuates wildly, even reaching 10^{-13}. However, the maximum relative error of all $b_m^{(N)}$ obtained by the XSpMV is less than 10^{-15}. Therefore, the accuracy and stability of the XSpMV algorithm are much better than that of the SpMV algorithm.

Furthermore, we select 50 sparse matrices from SuiteSparse Matrix Collection [57] randomly for testing. Figure 6 shows the maximum relative error results. It can be seen from Fig. 6 that the relative error generated by using the SpMV algorithm reaches a maximum of 10-12. Meanwhile, the relative error at all scales is more extensive than those produced by XSpMV and is very unstable.

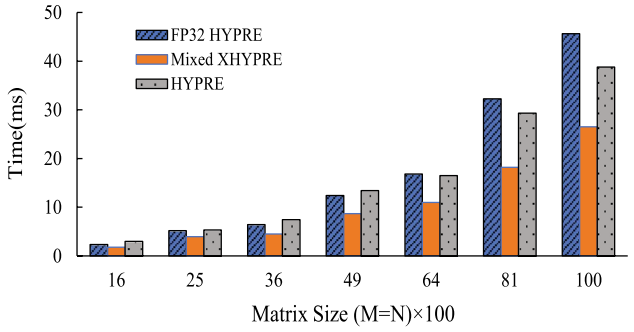

Fig. 10 The computation time of FP32 HYPRE, Mixed XHYPRE and HYPRE on the CPU

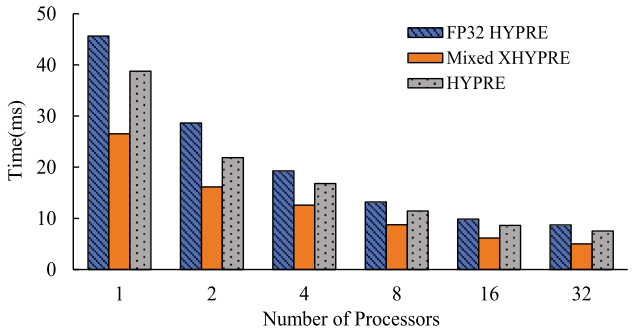

Fig. 11 The computation time of multi-processor FP32 HYPRE, Mixed XHYPRE and HYPRE on the CPU, matrix size is $M=N=1\times10^4$

The relative error generated by the XSpMV algorithm is smaller and more stable than that generated by SpMV. We continue to select 50 large-scale SuiteSparse matrices for experiments, as shown in Fig. 7. One can find that the XSpMV algorithm is very stable, and its relative errors are smaller than the working precision **u**, much less than the relative error of SpMV.

Finally, we test the calculation time of small-scale matrices and large-scale matrices. The running time of the two algorithms also gradually increases as the scale increases. However, the running time of the XSpMV algorithm is only slightly higher than that of the SpMV algorithm because the computational complexity (FLOPs) of the XSpMV is much higher than that of the SpMV. Moreover, the increase in precision will inevitably increase the running time of the program, and such a result is acceptable. This also shows that the XSpMV algorithm improves the accuracy of the classic SpMV algorithm while having a small impact on the overall calculation efficiency.

Table 5 Example informations

Name	Problem description	Convergence
ex4	The convection-reaction-diffusion problem	1e-6
ex5	The 2-D Laplacian problem	1e-7
ex7	The convection-reaction-diffusion problem	1e-6

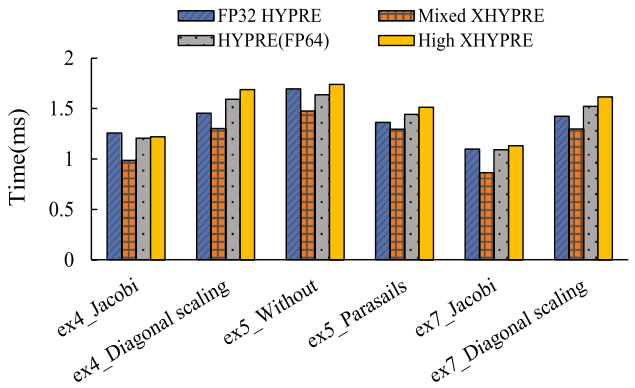

Fig. 12 The computation time of the different examples on the CPU

5.3 Performace analysis of high-precision XHYPRE

In this section, we use an example that both XHYPRE and HYPRE can solve for detailed experiments to illustrate the performance of our high-precision XHYPRE. We solve the 2-D Laplacian problem. The solver we use is PCG, and the convergence threshold is 1e-12.

Firstly, we test the number of iterations of high-precision XHYPRE and HYPRE under different matrix sizes to show the performance of our high-precision XHYPRE. Figure 8 shows the number of iterations of high-precision XHYPRE is less than that of the original HYPRE. As the matrix size increases, the iteration difference becomes more apparent. The number of iterations of XHYPRE is much smaller than that of HYPRE. But in order to obtain more accurate calculation results, we add many calculation steps to high-precision XHYPRE, which will increase the running time, as shown in Table 3. This is acceptable.

Then, we use multiple processors to parallel execute high-precision XHYPRE to test its performance. Figure 9 shows that the calculation time gradually decreases with the increase of processors, and the difference between High XHYPRE and HYPRE keeps shrinking. When the number of processors increases to 16, their time overhead is almost equal, because the reduction in the number of iterations reflects better performance under the current processor scale.

Therefore, our high-precision XHYPRE can better reduce the rounding error in the calculation process when solving large-scale linear problems, improve the accuracy of the global calculation, and obtain a lower number of iterations.

This is essential when solving ill-conditioned problems (see Sect. 4.1). Moreover, our XHYPRE can also show better performance when parallel solving.

5.4 Performace analysis of nixed-precision XHYPRE

In this section, we adopt an example where both XHYPRE and HYPRE can be solved to conduct sufficient experiments to illustrate the performance of our mixed-precision XHYPRE. The example and solver we use are the same as in Sect. 5.3), and the convergence threshold is 1e-6.

First, we test the number of iterations of single-precision HYPRE (FP32 HYPRE), mixed-precision XHYPRE (Mixed XHYPRE), and HYPRE for different matrix sizes. It can be clearly found from Table 4 that our mixed-precision XHYPRE has fewer iterations than single-precision HYPRE, which is basically the same as the original HYPRE.

Then, we further test their computation time on the CPU. The convergence speed of our mixed-precision XHYPRE is the fastest in Fig. 10. Because Mixed XHYPRE has fewer iterations compared with single-precision HYPRE, and Mixed XHYPRE has a lower working accuracy compared with the original HYPRE under the same number of iterations. Hence, the running time of Mixed XHYPRE is the least. In addition, we run parallel experiments with multiple processors, and the results are shown in Fig. 11. The calculation time of each HYPRE version is reducing with the increase of processors, but the running time of our mixed XHYPRE has always been the least than the other two. This is because we are running algorithms at reduced precision while maintaining high precision in crucial operations, and we use lower precision data formats as much as possible to achieve high computing accuracy. Therefore, our mixed-precision XHYPRE can overcome the poor convergence of FP32 HYPRE and the long computation time of FP64 HYPRE.

5.5 Overall performace of XHYPRE

5.5.1 Experimental analysis on different examples

In order to show the performance of our XHYPRE more comprehensively, we give a detailed description of the experiment in this section. The examples involved in the experiment are from HYPRE. We select one of the different examples that solve the same problem and use the same interface for testing.

Firstly, we selected single-precision HYPRE, mixed-precision HYPRE, double-precision HYPRE, and high-precision HYPRE for experiments. We use the PCG solver and different preconditioners to solve the same example. We set the matrix size as 1089×1089. The detailed information of the example is shown in Table 5. Although ex4 and ex7 solve the same problem, they use different interfaces. The comparison shows that the iterations are approximately consistent between our mixed-precision XHYPRE and double-precision HYPRE, with single-precision HYPRE needing more iterations.

Then, we test their computation time on the CPU, as shown in Fig. 12. Our mixed-precision XHYPRE has the fastest convergence speed. Because mixed-precision XHYPRE has fewer iterations than non-mixed FP32 HYPRE, it is faster than non-mixed FP32 HYPRE. Although the calculation time of High XHYPRE is slightly higher than the other three, it reduces the rounding error in the calculation and obtains a more accurate result, which is acceptable. Our mixed-precision XHYPRE can overcome the poor convergence of FP32 HYPRE and the long computation time of FP64 HYPRE. It can be used to solve large-scale scientific calculations when combined with specialized computer hardware.

From the above analysis, we can draw two conclusions. One is that mixed-precision XHYPRE can leverage the potential performance advantages of mixed precision, do most of the work in low-precision algorithms, successfully generate higher accuracy solutions, and increase convergence speed. The other is that high-precision XHYPRE improves the calculation accuracy by reducing the rounding

Table 6 Machine parameters

Machine	Parameters description
Intel	Intel(R) Xeon(R) CPU E5-2678 v3, @2.50GHz, 24T, 125 G memory
AMD	AMD Ryzen 7 2700X, @2.10 GHz, 16T, 64 G memory
Arm	FT-2000+, @2.2GHz, 16T, 124 G memory

Table 7 The computation time of different versions of HYPRE in Intel, AMD, and Arm

Name	Intel (ms)	AMD (ms)	Arm (ms)
FP32 HYPRE	40.174	44.686	48.655
Mixed XHYPRE	25.566	26.049	28.675
HYPRE(FP64)	35.180	38.117	40.963

Table 8 Iterations of HYPRE and high-precision HYPRE in Intel, AMD, and Arm

Name	Intel	AMD	Arm
HYPRE(FP64)	2698	2698	2698
High XHYPRE	2273	2273	2273

error for the problems with higher precision requirements and some ill-conditioned problems. Therefore, our XHYPRE achieves better performance.

5.5.2 Experimental analysis on different platforms

In this section, we conduct experiments on different platforms to verify the robustness and portability of XHYPRE. Table 6 shows more details of machine parameters. The example is the 2-D Laplacian problem, and the solver is PCG.

Firstly, we test the calculation time of our XHYPRE on Intel, AMD, and Arm, respectively. We set the matrix size as 10000×10000, and the convergence threshold is 1e-7. Table 7 shows that our XHYPRE is faster than other two versions, no matter for which platform. Then we test the number of iterations for high-precision XHYPRE in Intel, AMD, and Arm, respectively. We increase the matrix size by 100 times, and the convergence threshold is set to 1e-12. As shown in Table 8, our high-precision XHYPRE has fewer iterations, which improves the calculation accuracy. Therefore, our XHYPRE can run under various architectures while maintaining high performance, and it is an efficient and reliable parallel numerical algorithm library.

6 Conclusion and future work

The scale of scientific computing is gradually increasing on the current high-performance computing platform, and the requirements for precision are progressively growing. In this paper, we utilize EFT technology and mixed-precision idea to implement a reliable and efficient numerical algorithm library XHYPRE for large sparse linear equations. It gives full use to the fusion advantages of the algorithm and significantly improves the accuracy and efficiency of numerical calculations. Our XHYPRE involves two aspects. One is the high-precision XHYPRE, which uses EFT technology to reduce the accumulation of rounding errors, and designs reliable numerical algorithms to make the output results more accurate and stable. The other is the mixed-precision XHYPRE, which leverages mixed-precision to speed up calculations and improve overall performance. A large number of numerical experiments prove that XHYPRE effectively improves the accuracy and efficiency of computing. Our XHYPRE is on average 1.3x faster than HYPRE and reduces the number of iterations to 87.1% on average. Besides, the proposed XHYPRE can solve the ill-conditioned problems that HYPRE and PETSc can't solve.

The increase in precision will also reduce the performance because more floating-point calculations are required. In addition, if the computing scale is large enough, it will increase a lot of time costs. XHYPRE will be compiled and optimized in future work to reduce the rounding error and the running time. Based on current research, we will combine EFT and mixed-precision thought to add other functions to make XHYPRE more comprehensive. Furthermore, we also plan to implement our library on the GPU to achieve higher performance.

Acknowledgements This work was supported by the NuSCAP (ANR-20-CE48-0014) project of the French National Agency for Research (ANR), the 173 program (2020-JCJQ-ZD-029), Science Challenge Project (TZ2016002).

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no confict of interest.

References

Abdelfattah, A., Anzt, H., Boman, E.G., Carson, E., Cojean, T., Dongarra, J., Fox, A., Gates, M., Higham, N.J., Li, X.S., et al.: A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. Int. J. High Perform. Comput. Appl. **35**(4), 344–369 (2021). https://doi.org/10.1177/10943420211003313

Abdulah, S., Cao, Q., Pei, Y., Bosilca, G., Dongarra, J., Genton, M.G., Keyes, D.E., Ltaief, H., Sun, Y.: Accelerating geostatistical modeling and prediction with mixed-precision computations: a high-productivity approach with parsec. IEEE Trans. Parallel Distrib. Syst. **33**(4), 964–976 (2022). https://doi.org/10.1109/TPDS.2021.3084071

Baboulin, M., Buttari, A., Dongarra, J., Kurzak, J., Langou, J., Langou, J., Luszczek, P., Tomov, S.: Accelerating scientific computations with mixed precision algorithms. Comput. Phys. Commun. **180**(12), 2526–2533 (2009). https://doi.org/10.1016/j.cpc.2008.11.005

Bailey, D.H., Barrio, R., Borwein, J.M.: High-precision computation: Mathematical physics and dynamics. Appl. Math. Comput. **218**(20), 10106–10121 (2012). https://doi.org/10.1016/j.amc.2012.03.087

Baker, A.H., Falgout, R.D., Kolev, T.V., Yang, U.M.: Scaling hypre's multigrid solvers to 100,000 cores. High-Perform. Sci. Comput. (2012). https://doi.org/10.1007/978-1-4471-2437-5_13

Benz, F., Hildebrandt, A., Hack, S.: A dynamic program analysis to find floating-point accuracy problems. ACM SIGPLAN Not. **47**(6), 453–462 (2012). https://doi.org/10.1145/2345156.2254118

Blanchard, P., Higham, N.J., Lopez, F., Mary, T., Pranesh, S.: Mixed precision block fused multiply-add: error analysis and application to gpu tensor cores. SIAM J. Sci. Comput. **42**(3), 124–141 (2020). https://doi.org/10.1137/19M1289546

Carson, E., Higham, N.J.: Accelerating the solution of linear systems by iterative refinement in three precisions. SIAM J. Sci. Comput. **40**(2), 817–847 (2018). https://doi.org/10.1137/17M1140819

Connolly, M.P., Higham, N.J., Mary, T.: Stochastic rounding and its probabilistic backward error analysis. SIAM J. Sci. Comput. **43**(1), 566–585 (2021). https://doi.org/10.1137/20M1334796

Cools, S., Yetkin, E.F., Agullo, E., Giraud, L., Vanroose, W.: Analyzing the effect of local rounding error propagation on the maximal attainable accuracy of the pipelined conjugate gradient method. SIAM J. Matrix Anal. Appl. **39**(1), 426–450 (2018). https://doi.org/10.1137/17M1117872

de Camargo, A.P.: On the numerical stability of newton's formula for lagrange interpolation. J. Comput. Appl. Math. **365**, 112369 (2020). https://doi.org/10.1016/j.cam.2019.112369

Dekker, T.J.: A floating-point technique for extending the available precision. Numerische Mathematik **18**(3), 224–242 (1971). https://doi.org/10.1137/030601818

Delgado Gracia, J.: Compensated evaluation of tensor product surfaces in cagd. Mathematics **8**(12), 2219 (2020). https://doi.org/10.3390/math8122219

Du, P., Barrio, R., Jiang, H., Cheng, L.: Accurate quotient-difference algorithm: error analysis, improvements and applications. Appl. Math. Comput. **309**, 245–271 (2017). https://doi.org/10.1016/j.amc.2017.04.004

Engwer, C., Falgout, R.D., Yang, U.M.: Stencil computations for pde-based applications with examples from dune and hypre. Concurr. Comput.: Pract. Exp. **29**(17), 4097 (2017). https://doi.org/10.1002/cpe.4097

Falgout, R.D., Yang, U.M.: hypre: a library of high performance preconditioners. Int. Conf. Comput. Sci. (2002). https://doi.org/10.1007/3-540-47789-6_66

Falgout, R.D., Jones, J.E., Yang, U.M.: The design and implementation of hypre, a library of parallel high performance preconditioners. Numer. Solut. Partial Diff. Equ. Parallel Comput. (2006). https://doi.org/10.1007/3-540-31619-1_8

Falgout, R.D., Jones, J.E., Yang, U.M.: Conceptual interfaces in hypre. Futur. Gener. Comput. Syst. **22**(1–2), 239–251 (2006). https://doi.org/10.1016/j.future.2003.09.006

Gershman, R., Strichman, O.: Cost-effective hyper-resolution for preprocessing cnf formulas. In: International Conference on Theory and Applications of Satisfiability Testing, pp. 423–429 (2005). https://doi.org/10.1007/11499107_34

Graillat, S., Ménissier-Morain, V.: Compensated horner scheme in complex floating point arithmetic. In: Proceedings of the 8th Conference on Real Numbers and Computers, Santiago de Compostela, Spain, pp. 133–146 (2008)

Graillat, S., Jézéquel, F.: Tight interval inclusions with compensated algorithms. IEEE Trans. Comput. **69**(12), 1774–1783 (2020). https://doi.org/10.1109/TC.2019.2924005

Graillat, S., Jézéquel, F., Picot, R.: Numerical validation of compensated algorithms with stochastic arithmetic. Appl. Math. Comput. **329**, 339–363 (2018). https://doi.org/10.1016/j.amc.2018.02.004

Haidar, A., Bayraktar, H., Tomov, S., Dongarra, J., Higham, N.J.: Mixed-precision iterative refinement using tensor cores on gpus to accelerate solution of linear systems. Proc. R. Soc. A **476**(2243), 20200110 (2020). https://doi.org/10.1098/rspa.2020.0110

Hermes, D.: Compensated de casteljau algorithm in k times the working precision. Appl. Math. Comput. **357**, 57–74 (2019). https://doi.org/10.1016/j.amc.2019.03.047

Higham, N.J., Mary, T.: A new approach to probabilistic rounding error analysis. SIAM J. Sci. Comput. **41**(5), 2815–2835 (2019). https://doi.org/10.1137/18M1226312

https://github.com/solverchallenge/solverchallenge21-tenproblems

https://sparse.tamu.edu/

https://www.mcs.anl.gov/petsc/

Hypre:https://computing.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods

Jiang, H., Graillat, S., Hu, C., Li, S., Liao, X., Cheng, L., Su, F.: Accurate evaluation of the k-th derivative of a polynomial and its application. J. Comput. Appl. Math. **243**, 28–47 (2013). https://doi.org/10.1016/j.cam.2012.11.008

Jin, G., Mellor-Crummey, J.: Experiences tuning smg98: a semicoarsening multigrid benchmark based on the hypre library. Proc. 16th Int. Conf. Supercomput. (2002). https://doi.org/10.1145/514191.514233

Knuth, D.E.: Art of Computer Programming, Volume 2: Seminumerical Algorithms, (2014)

Knyazev, A.V., Argentati, M.E., Lashuk, I., Ovtchinnikov, E.E.: Block locally optimal preconditioned eigenvalue xolvers (blopex) in hypre and petsc. SIAM J. Sci. Comput. **29**(5), 2224–2239 (2007). https://doi.org/10.1137/060661624

Kurzak, J., Buttari, A., Dongarra, J.: Solving systems of linear equations on the cell processor using cholesky factorization. IEEE Trans. Parallel Distrib. Syst. **19**(9), 1175–1186 (2008). https://doi.org/10.1109/TPDS.2007.70813

Lashuk, I., Argentati, M., Ovtchinnikov, E., Knyazev, A.: Preconditioned eigensolver lobpcg in hypre and petsc. Domain Decompos. Methods Sci. Eng. **16**, 635–642 (2007). https://doi.org/10.1007/978-3-540-34469-8_79

Li, C., Xiao, X., Du, P., Jiang, H., Barrio, R., Quan, Z., Li, K.: A high-precision dqds algorithm. In: 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), pp. 633–639 (2021). IEEE

Li, C., Du, P., Li, K., Liu, Y., Jiang, H., Quan, Z.: Accurate goertzel algorithm: error analysis, validations and applications. Mathematics **10**(11), 1788 (2022)

Li, C., Barrio, R., Xiao, X., Du, P., Jiang, H., Quan, Z., Li, K.: Pacf: A precision-adjustable computational framework for solving singular values. Appl. Math. Comput. **440**, 127611 (2023). https://doi.org/10.1016/j.amc.2022.127611

Lindquist, N., Luszczek, P., Dongarra, J.: Accelerating restarted gmres with mixed precision arithmetic. IEEE Trans. Parallel Distrib. Syst. **33**(4), 1027–1037 (2022). https://doi.org/10.1109/TPDS.2021.3090757

Mascarenhas, W.F., de Camargo, A.P.: The effects of rounding errors in the nodes on barycentric interpolation. Numerische Mathematik **135**(1), 113–141 (2017). https://doi.org/10.1007/s00211-016-0798-x

McCormick, S.F., Benzaken, J., Tamstorf, R.: Algebraic error analysis for mixed-precision multigrid solvers. SIAM J. Sci. Comput. **43**(5), 392–419 (2021). https://doi.org/10.1137/20M1348571

Menon, H., Lam, M.O., Osei-Kuffuor, D., Schordan, M., Lloyd, S., Mohror, K., Hittinger, J.: Adapt: Algorithmic differentiation applied to floating-point precision tuning. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 614–626 (2018). https://doi.org/10.1109/SC.2018.00051

Muller, J.-M., Brisebarre, N., de Dinechin, F., Jeannerod, C.-P., Lefèvre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S.: The Fused Multiply-Add Instruction, pp. 151–179. Birkhäuser Boston, Boston (2010). https://doi.org/10.1007/978-0-8176-4705-6_5

Muller, J.-M., Brisebarre, N., De Dinechin, F., Jeannerod, C.-P., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S., et al.: Handbook of floating-point Arithmetic. Birkhauser (2018)

Ogita, T., Rump, S.M., Oishi, S.: Accurate sum and dot product. SIAM J. Sci. Comput. **26**(6), 1955–1988 (2005). https://doi.org/10.1137/030601818

Ozaki, K., Terao, T., Ogita, T., Katagiri, T.: Verified numerical computations for large-scale linear systems. Appl. Math. **66**(2), 269–285 (2021)

Petschow, M., Quintana-Ortí, E.S., Bientinesi, P.: Improved accuracy and parallelism for mrrr-based eigensolvers–a mixed precision approach. SIAM J. Sci. Comput. **36**(2), 240–263 (2014). https://doi.org/10.1137/130911561

Sahasrabudhe, D., Berzins, M.: Improving performance of the hypre iterative solver for uintah combustion codes on manycore architectures using mpi endpoints and kernel consolidation. Int. Conf. Comput. Sci. (2020). https://doi.org/10.1007/978-3-030-50371-0_13

Sahasrabudhe, D., Zambre, R., Chandramowlishwaran, A., Berzins, M.: Optimizing the hypre solver for manycore and gpu architectures. J. Comput. Sci. **49**, 101279 (2021). https://doi.org/10.1016/j.jocs.2020.101279

Schmidt, J., Berzins, M., Thornock, J., Saad, T., Sutherland, J.: Large scale parallel solution of incompressible flow problems using uintah and hypre. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, pp. 458–465 (2013). https://doi.org/10.1109/CCGrid.2013.10

Sorna, A., Cheng, X., D'Azevedo, E., Won, K., Tomov, S.: Optimizing the fast fourier transform using mixed precision on tensor core hardware. In: 2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW), pp. 3–7 (2018). https://doi.org/10.1109/HiPCW.2018.8634417

Stummel, F.: Rounding error analysis of elementary numerical algorithms. Fundam. Numer. Comput. (computer-oriented numerical analysis) (1980). https://doi.org/10.1007/978-3-7091-8577-3_13

Sun, J., Peterson, G.D., Storaasli, O.O.: High-performance mixed-precision linear solver for fpgas. IEEE Trans. Comput. **57**(12), 1614–1623 (2008). https://doi.org/10.1109/TC.2008.89

Tan, G., Shui, C., Wang, Y., Yu, X., Yan, Y.: Optimizing the linpack algorithm for large-scale pcie-based cpu-gpu heterogeneous systems. IEEE Trans. Parallel Distrib. Syst. **32**(9), 2367–2380 (2021). https://doi.org/10.1109/TPDS.2021.3067731

Wei, J., Chen, M., Wang, L., Ren, P., Lei, Y., Qu, Y., Jiang, Q., Dong, X., Wu, W., Wang, Q., et al.: Status, challenges and trends of data-intensive supercomputing. CCF Trans. High. Perform. Comput. (2022). https://doi.org/10.1007/s42514-022-00109-9

Yang, W., Li, K., Li, K.: A hybrid computing method of spmv on cpu-gpu heterogeneous computing systems. J. Parallel Distrib. Comput. **104**, 49–60 (2017). https://doi.org/10.1016/j.jpdc.2016.12.023

Zhang, L., Gong, X., Song, J., Hu, J.: Parallel preconditioned gmres solvers for 3-d helmholtz equations in regional non-hydrostatic atmosphere model. 2008 Int. Conf. Comput.Sci. Softw. Eng. **3**, 287–290 (2008). https://doi.org/10.1109/CSSE.2008.898

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.