A General Framework for Reproducible Parallel Preconditioned Krylov Methods Using Three BLAS Variants

xiaojun Lei*
Graduate School of Chinese Academy of Engineering Physics
Beijing, Beijing, China
3292694852@qq.com

Tongxiang Gu
Institute of Applied Physics and Computational Mathematics
Beijing, Beijing, China
txgu@iapcm.ac.cn

Xiaowen Xu
Institute of Applied Physics and Computational Mathematics
Beijing, Beijing, China
xwxu@iapcm.ac.cn

Stef Graillat
Sorbonne Universite, CNRS, LIP6
Paris, Paris, France
stef.graillat@lip6.fr

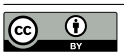
Abstract

Krylov subspace methods are crucial techniques for solving linear systems. To effectively tackle large-scale linear systems, parallelism techniques are frequently utilized. However, the use of parallelism can increase the non-associativity of floating-point operations, potentially resulting in non-reproducibility of the computations. In order to address the issue of non-reproducibility in parallel preconditioned Krylov subspace iterative methods, we investigated the sources of irreproducibility and solutions to achieve reproducibility. The main reason for the irreproducibility in iterative methods is attributed to distributed dot products. To tackle this, we investigated three reproducible Basic Linear Algebra Subprograms (BLAS) as replacements for standard distributed dot products. We utilized OzBLAS, ExBLAS, and ReproBLAS to ensure reproducibility in iterative methods. We combined Jacobi preconditioners and tested and verified on three iterative methods: CG, BiCGSTAB and GMRES. Numerical experiments demonstrate that the iterative method based on OzBLAS is reproducible in a multi-threaded environment. The iterative method based on ExBLAS and ReproBLAS achieves reproducibility in a multi-process environment. Our evaluation indicated that ReproBLAS performed well in terms of performance, robustness, and parallelism. Therefore, we employ a reproducible iterative method based on ReproBLAS to solve the large-scale algebraic equations resulting from the discretization of the two-dimensional three-temperature heat conduction equation, which arises from practical application problems in the field of laser fusion radiation hydrodynamics. Numerical experiments demonstrate the reproducible solution of the three-temperature heat conduction algebraic equations.

CCS Concepts

• **Mathematics of computing → Solvers**.

*Corresponding author



HP3C 2025, Jinan, China

ACM ISBN 979-8-4007-1853-3/25/06
https://doi.org/10.1145/3774949.3774958

Keywords

reproducibility, OzBLAS, ExBLAS, ReproBLAS, Krylov subspace iterative methods

ACM Reference Format:
xiaojun Lei, Tongxiang Gu, Xiaowen Xu, and Stef Graillat. 2025. A General Framework for Reproducible Parallel Preconditioned Krylov Methods Using Three BLAS Variants. In *2025 9th International Conference on High Performance Compilation, Computing and Communications (HP3C) (HP3C 2025), June 18–20, 2025, Jinan, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3774949.3774958

1 Introduction

In fields like laser fusion, fluid dynamics, structural analysis, biomedicine, and electronics, partial differential equations are fundamental for modeling complex systems. To solve them numerically, they must be discretized into linear equations. Efficiently solving these linear equations is crucial for accurate simulations. For the large, sparse matrices that arise in practical scenarios, iterative methods rooted in Krylov subspace are typically more efficient, as they offer reduced computational complexity compared to direct solvers when addressing large-scale challenges. However, the straightforward application of iterative methods may lead to inadequate or slow convergence rates. To mitigate this issue, preconditioning techniques are extensively employed to enhance the convergence properties of the iterative process and to optimize computational efficiency. By utilizing a well-designed preconditioner, the solution process can be greatly expedited, rendering the Krylov subspace iterative methods more effective and practical for complex, large-scale numerical simulations.

According to the different selections of constraint spaces [14], Krylov subspace methods are divided into three categories. The first category belongs to orthogonal projection, and the corresponding methods are called orthogonal projection methods. This type of method can minimize the energy norm of the error. The representative and commonly used method is the CG [2]. The second category belongs to oblique projection, and the corresponding methods are called orthogonalization methods. This type of method can minimize the 2-norm of the residual. The representative and commonly used method is the GMRES method [2]. The third type of method

is called biorthogonalization method, and the representative and commonly used method is the BiCGSTAB [2].

In the field of scientific computing, reproducibility is very important. Non-reproducible behavior will cause verification and debugging problems, and may even lead to deadlock [4]. In our previous work, the hybrid solver project faced reproducibility issues due to inconsistent execution order in OpenMP's Allreduce, resulting in variations in backward errors and iterations. Besides the reproducibility issues we encountered, other literature also emphasizes this topic, such as [15]. Achieving bitwise identical floating-point results across multiple runs is essential for debugging and verifying correctness. One might expect that both serial and parallel executions of preconditioned iterative methods would produce identical outcomes, including iterations, residuals, and resulting vectors. However, this is often not the case due to factors like different reduction trees—MPI offers 14 implementations—data alignment, specific instructions, and input data sequences, all of which affect the order of floating-point operations. Since floating-point addition is commutative but not associative, these variations can lead to non-reproducible results. Our goal is to develop reproducible parallel preconditioned iterative algorithms that yield the same results with varying process counts on the same platform. However, current research on reproducibility is mainly at the BLAS level, such as OzBLAS, ExBLAS and ReproBLAS. In this paper, We will analyze the sources of non-reproducibility in the iterative method and present a general framework for reproducible parallel preconditioned Krylov methods.

Reproducibility in this paper means that parallel programs use different numbers of processes to obtain the same results bit by bit. More generally, reproducibility also includes obtaining the same results bit by bit on different hardware platforms. More specifically, this paper studies how to use a parallel solver to solve a system of linear algebraic equations to ensure that serial execution and parallel execution with different numbers of processes or threads produce the same results bit by bit. Reproducibility and accuracy problems have different motivations, but both stem from rounding errors. While improving reproducibility doesn't increase accuracy, enhancing accuracy can help reduce reproducibility issues.

The rest of the paper is organized as follows. Section 2 introduces three types of BLAS: OzBLAS, ExBLAS, and ReproBLAS. Section 3 introduces three types of Krylov subspace iterative methods and their MPI implementation details. We present strategies for ensuring reproducibility in section 4. Section 5 conducts extensive numerical experiments to demonstrate the reproducibility, performance, and accuracy of different BLAS, as well as the reproducibility of large-scale matrices. Section 6 introduces some related work. Finally, Section 7 draws conclusions and proposes future work.

2 Three BLAS

2.1 OzBLAS based on error free transformation

OzBLAS [10] is a reproducible BLAS library based on thread level operations (Ozaki et al., 2012). It has already been implemented on x86 multi-core CPU and GPU. It is reproducible in a multi-threaded environment. The library also implements the CG iterative method. The Ozaki scheme [12] is an error free transformation of dot product/matrix multiplication. This technique can be seen as a broadening of high-precision arithmetic approaches, such as double-double precision, to incorporate multiple components at the vector level. By aggregating the pairwise inner products of segmented vectors, it ensures the production of accurate and reproducible outcomes. They use the correct rounding method NearSum [13] to calculate the sum. Therefore, its computation results are reproducible and of high precision, with the relative error of the computed dot product reaching machine precision. This library can be found in https://www.r-ccs.riken.jp/labs/lpnctrt/projects/ozblas/.

2.2 ExBLAS adapted to multi-level architecture

The ExBLAS project [6] is an attempt to build a fast, accurate, and reproducible BLAS library for various modern architectures. The project utilizes a multi-level strategy to execute these operations, effectively adapting to the intricate multi-level memory architectures of contemporary systems. ExBLAS integrates long accumulators and floating-point extensions (FPE) into its algorithmic framework. The method boasts two distinct features. Firstly, it is designed to preserve every bit of information prior to rounding to the target format, ensuring accuracy and correct rounding. As a result, the computation yields reproducible outcomes with high precision, where the relative error of the computed dot product approaches machine precision. Secondly, it has been meticulously optimized and implemented across a variety of architectures, including traditional CPUs, Nvidia GPUs, AMD GPUs, and Intel Xeon Phi coprocessors. This library can be found in https://github.com/riakymch/exblas.

2.3 ReproBLAS based on reproducible summation

ReproBLAS [1] aims to offer users a suite of basic linear algebra subroutines, both parallel and sequential, designed to ensure the reproducibility of results. This consistency is maintained irrespective of the number of processors used, the methods of data partitioning, the scheduling of reductions, and the overall sequence of calculations and summations. The foundation of ReproBLAS is its reproducible summation, which remains consistent regardless of the summation order. In sequential processing, the algorithm traverses the data a single time. In parallel scenarios, it requires only one reduction operation, utilizing an accumulator that consists of 6 floating-point numbers. Additional floating-point numbers may be employed to achieve greater accuracy if needed. Using an accumulator with 6 floating-point numbers incurs an arithmetic cost of performing $7n$ floating-point additions for n numbers. Under IEEE double precision standards, the final error bound can be up to 10^{-8} times smaller than that of the traditional sum error bound. Let $T = \sum_{j=0}^{n-1} x_j$, $\overline{T}$ be the floating-point approximate sum obtained using ReproBLAS. The absolute error of the reproducible summation algorithm is $|T - \overline{T}| \leq n \times 2^{W(1-K)} \max |x_j| + 7\mathrm{u}| \sum_{j=0}^{n-1} x_j|$, where u is the unit rounding error. By default, the parameters for double precision are $W = 40$ and $K = 3$. This library can be obtained at the following link: https://bebop.cs.berkeley.edu/reproblas/.

3 Preconditioned Krylov Methods

3.1 Preconditioned iteration method

The most common preconditioned Krylov subspace iterative methods are preconditioned Conjugate Gradient (PCG), preconditioned GMRES, and preconditioned BiCGSTAB. Let's take these three algorithms, preconditioned Conjugate Gradient (PCG), preconditioned GMRES, and preconditioned BiCGSTAB, as examples to discuss how to achieve reproducibility in preconditioned iterative methods.

3.1.1 Preconditioned CG. The CG algorithm is suitable for solving symmetric positive definite problems and minimizes the error's energy norm. The algorithm flow can be found in [2].

3.1.2 Preconditioned GMRES. The GMRES algorithm is optimal in the sense of minimizing the residual 2-norm, but it is based on long recursions. As the number of iterations increases, both computation and storage requirements grow linearly. The algorithm flow can be found in [2].

3.1.3 Preconditioned BiCGSTAB. The BICGSTAB algorithm only achieves local optimality, but it is based on short recursions. The algorithm flow can be found in [2].

3.2 Parallel Preconditioned Iterative Method

We take the example of the preconditioned BiCGSTAB algorithm to illustrate how to implement a parallel preconditioned iteration method. The parallel implementation of preconditioned CG and preconditioned GMRES is very similar to the parallel implementation of preconditioned BiCGSTAB. In our parallel preconditioned BiCGSTAB algorithm, we utilize L processes where the matrix A is partitioned into L row blocks $(A_1, A_2, \ldots, A_i, \ldots, A_L)$. Each process is responsible for storing and managing one of these row blocks. Correspondingly, the vectors are distributed in alignment with the row blocks of A. The SpMV operations necessitate using MPI_Allgatherv(). This MPI function is employed to aggregate the local vector segments from each process to form a complete vector necessary for the operation. Moreover, the algorithm requires the calculation of distributed dot products. For the non-reproducible version of the parallel preconditioned BiCGSTAB, we utilize MPI_Allreduce() to perform these operations. However, for the reproducible version, we implement specialized reproducible distributed dot product computations, which differ from the standard approach. This distinction in handling dot products forms the primary difference between the reproducible and non-reproducible versions of the parallel preconditioned BiCGSTAB algorithm. For detailed information on reproducible distributed dot products, please refer to sections 4.

4 Strategies for Reproducibility

We take the parallel preconditioned BiCGSTAB algorithm as an example to illustrate how to achieve reproducibility. The parallel preconditioned CG algorithm and parallel preconditioned GMRES algorithm achieve reproducibility in a very similar way to the parallel preconditioned BiCGSTAB algorithm. In this section, we detail our approach for ensuring reproducibility in the parallel preconditioned BiCGSTAB algorithm. We identify the elements within the PBiCGSTAB solver that contribute to nondeterministic computations and present our proposed solutions to address these issues.

Dot products: The primary issue of non-determinism arises from dot products and parallel reduction operations, specifically MPI_Allreduce(). We use the ExBLAS and OzBLAS method to provide reproducible and correctly rounded dot products, and the ReproBLAS method to provide reproducible dot products.

Sparse matrix-vector product: Each process possesses a local matrix A_i, and the complete vector is assembled using MPI_Allgatherv(). Subsequently, each process performs a SpMV. Since these computations are carried out locally and sequentially, they are deterministic and, therefore, reproducible.

AXPY(-type) vector updates: For computing the axpy(-type) vector updates, each processor independently updates its local vector. These operations are deterministic and, therefore, reproducible.

Application of the preconditioner: We employ a Jacobi preconditioner, whose application is straightforward. Initially, the inverses of the diagonal elements are computed. Subsequently, applying the preconditioner simply involves an element-wise multiplication of two vectors. Consequently, this process is both correctly rounded and reproducible. For other preconditioners, reproducibility needs to be achieved based on their characteristics.

Table 1: Test matrices.

Matrix	n	nnz	kind
cfd1	70, 656	1, 825, 580	Computational Fluid Dynamics Problem
gridgena	48, 962	512, 084	optimization problem
s1rmq4m1	5, 489	262, 411	Structural Problem

5 Numerical Results

5.1 Setup

The experiments were conducted on the Sugon HPC cluster, which includes 172 compute nodes. Each node is equipped with two 12-core Intel E5-2680 v3 processors, totaling 24 cores per node. The MPI library used for these experiments is MPICH.

In sections 5.2-5.3, we selected three matrices from SuiteSparse Matrix Collection [3] to test reproducibility and performance, and the matrix information is shown in Table 1. In sections 5.4, we use Algorithm 4.2 in [16] to generate ill-conditioned dot product data to test the accuracy of different BLAS. In sections 5.5, We select large-scale matrices from real applications to test reproducible iterative methods.

The right-hand side vector b in the iterative solvers was always initialized to the product $A(1, 1, \ldots, 1)^T$, and the initial guess $x^{(0)} = 0$. The stopping criterion is $\|r^{(j)}\|_2 \leq 10^{-8}$, where j is the number of iterations.

In the experimental results shown below, OzBLAS used one node with 24 threads bound to one core, while Regular (MPI_Allreduce), ExBLAS and ReproBLAS used one node with 24 processes bound to one core. The parameter ReproBLAS to control precision was set to $K = 3$.

To reproduce the numerical experiments of this study, the code can be found at https://github.com/programmer-lxj/Reproducible-Parallel-Preconditioned-Krylov-Methods-Based-on-Three-Different-BLAS.

Table 2: Number of iterations of PCG.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	1824	2760	1823	1823
gridgena	3833	2583	3833	3833
s1rmq4m1	905	10000	904	905

Table 3: Number of iterations of PGMRES.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	1601	1601	1601	1601
gridgena	4110	4104	4104	4104
s1rmq4m1	8404	8393	8386	8374

Table 4: Number of iterations of PBiCGSTAB.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	1623	1331	1309	1492
gridgena	4410	2793	3373	3589
s1rmq4m1	839	702	757	702

Table 5: Final residuals of PCG.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$9.975e-9$	$5.820e-15$	$9.832e-9$	$9.958e-9$
gridgena	$9.840e-9$	$4.417e-15$	$9.807e-9$	$9.823e-9$
s1rmq4m1	$9.734e-9$	$1.795e-9$	$9.696e-9$	$8.216e-9$

5.2 Accuracy and Reproducibility Evaluation

In this section, we report the results of the accuracy and reproducibility evaluation. Hereafter, we call them OzBLAS, ExBLAS and ReproBLAS for short. Table 2, Table 3, and Table 4 respectively show the number of iterations required for convergence criteria to be met by PCG, PGMRES, and PBiCGSTAB.

Table 5, Table 6, and Table 7 respectively display the 2-norm of the final residuals for PCG, PGMRES, and PBiCGSTAB.

Table 8, Table 9, and Table 10 respectively show the infinity norm error between the approximate and exact solutions for PCG, PGMRES, and PBiCGSTAB upon meeting the convergence criteria.

Using different thread counts (8, 12, and 20), the errors of the three iterative methods based on OzBLAS, as well as ExBLAS and ReproBLAS with process counts of 8, 12, and 20, are all bitwise identical to the corresponding results in the tables for iterations, final residuals, and infinite norm errors. That is to say, the iterative method based on OzBLAS achieves reproducibility in a multi-threaded environment, while the iterative method based on ExBLAS and ReproBLAS achieves reproducibility in a multi-process environment.

5.3 Performance Evaluation

Table 11 shows the total time taken by PCG to converge, Table 12 shows the time for PGMRES, and Table 13 shows the time for PBiCGSTAB.

Table 14 presents the average time per iteration for PCG, Table 15 for PGMRES, and Table 16 for PBiCGSTAB.

Based on the average time per iteration as shown in Tables 14, 15, 16, the order from fastest to slowest on average is ReproBLAS, Regular, OzBLAS, and ExBLAS.

Table 6: Final residuals of PGMRES.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$9.922e-9$	$9.922e-9$	$9.922e-09$	$9.922e-9$
gridgena	$9.968e-9$	$9.906e-9$	$9.906e-9$	$9.906e-9$
s1rmq4m1	$9.802e-9$	$9.679e-9$	$9.998e-9$	$8.601e-9$

Table 7: Final residuals of PBiCGSTAB.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$5.881e-09$	$8.056e-09$	$7.208e-09$	$3.809e-09$
gridgena	$8.068e-09$	$9.379e-09$	$5.279e-09$	$6.957e-09$
s1rmq4m1	$2.405e-09$	$9.518e-09$	$3.803e-09$	$7.780e-09$

Table 8: Direct error of PCG.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$5.370e-8$	$2.576e-14$	$5.422e-8$	$5.373e-8$
gridgena	$1.156e-10$	$3.508e-14$	$1.156e-10$	$1.155e-10$
s1rmq4m1	$5.020e-12$	$3.235e-12$	$5.419e-12$	$4.759e-12$

Table 9: Direct error of PGMRES.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$7.153e-7$	$7.153e-7$	$7.153e-07$	$7.153e-7$
gridgena	$1.434e-9$	$1.520e-9$	$1.521e-9$	$1.524e-9$
s1rmq4m1	$7.680e-7$	$9.752e-7$	$8.549e-7$	$9.015e-7$

Table 10: Direct error of PBiCGSTAB.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$1.736e-06$	$8.870e-07$	$2.072e-06$	$4.065e-06$
gridgena	$5.659e-09$	$2.478e-09$	$6.499e-09$	$1.471e-08$
s1rmq4m1	$6.957e-11$	$1.318e-10$	$7.226e-11$	$1.102e-09$

Table 11: Total execution time until convergence, and execution time overhead of PCG.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$1.753e+00$	$1.317e+01$	$1.382e+01$	$1.713e+00$
gridgena	$1.590e+00$	$5.969e+00$	$1.957e+01$	$1.431e+00$
s1rmq4m1	$1.527e-01$	$8.672e+00$	$6.170e-01$	$1.447e-01$

Table 12: Total execution time until convergence, and execution time overhead of PGMRES.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$4.441e+01$	$6.750e+02$	$3.103e+03$	$9.450e+01$
gridgena	$5.218e+02$	$3.729e+03$	$1.357e+04$	$2.204e+02$
s1rmq4m1	$3.817e+02$	$8.146e+03$	$7.369e+03$	$4.319e+02$

Table 13: Total execution time until convergence, and execution time overhead of PBiCGSTAB.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$3.125e+00$	$1.368e+01$	$2.024e+01$	$2.947e+00$
gridgena	$3.518e+00$	$1.386e+01$	$3.201e+01$	$2.701e+00$
s1rmq4m1	$2.482e-01$	$1.827e+00$	$1.157e+00$	$2.138e-01$

Table 14: Average time per iteration of PCG.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$9.609e-04$	$4.688e-03$	$7.581e-03$	$9.394e-04$
gridgena	$4.149e-04$	$2.225e-03$	$5.106e-03$	$3.733e-04$
s1rmq4m1	$1.688e-04$	$8.454e-04$	$6.825e-04$	$1.599e-04$
average	$4.030e-4$	$2.586e-3$	$4.456e-3$	$3.807e-4$

Table 15: Average time per iteration of PGMRES.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$2.772e-02$	$4.214e-01$	$1.937e+00$	$5.899e-02$
gridgena	$1.269e-01$	$9.085e-01$	$3.306e+00$	$1.524e-02$
s1rmq4m1	$4.541e-02$	$9.706e-01$	$8.787e-01$	$5.157e-02$
average	$5.364e-2$	$5.792e-1$	$1.569e+0$	$3.399e-2$

Table 16: Average time per iteration of PBiCGSTAB.

Matrix	Regular	OzBLAS	ExBLAS	ReproBLAS
cfd1	$1.925e-3$	$1.027e-3$	$1.546e-2$	$1.975e-3$
gridgena	$7.977e-4$	$4.962e-4$	$9.490e-3$	$7.525e-4$
s1rmq4m1	$2.961e-4$	$2.602e-3$	$1.528e-3$	$3.045e-4$
average	$7.776e-4$	$1.765e-3$	$6.804e-3$	$7.822e-4$

5.4 Comparison of Three BLAS Accuracies

We utilize Algorithm 4.2 from Yamanaka et al. [16] to generate arbitrarily ill-conditioned dot product data. The size of the dot product data is $n = 2400$. The condition number of the dot product data generated by Algorithm 4.2 varies from 10^{10} to 10^{90}, where the exact result of the dot product is equal to cond^{-1}.

Fig.1 is a comparison of the accuracy of three BLAS and recursive calculations of dot products. The horizontal axis represents the condition number of the dot product problem, and the vertical axis represents the relative error. From the Fig.1, we can observe that the precision from high to low is OzBLAS, ExBLAS, ReproBLAS, and Regular. OzBLAS and ExBLAS have almost the same precision, with the relative error in computing the dot product reaching machine precision.

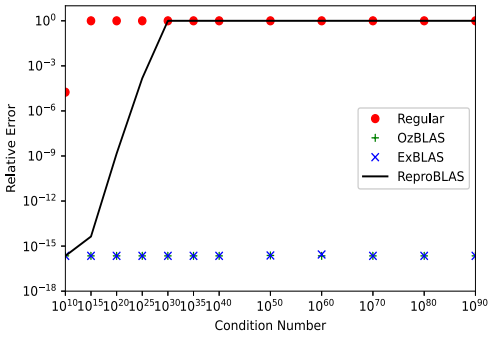


Figure 1: Comparison of the accuracy of three BLAS calculations of dot products.

5.5 Large scale matrices in practical applications

This section applies the reproducible iterative method to solve the linear system derived from the discretized three-dimensional three-temperature heat conduction equation, relevant to laser fusion radiation hydrodynamics. It involves structured mesh with a grid size of $128 \times 128 \times 128$, with 3 degrees of freedom per grid. Discretization is performed using a finite volume 7-point scheme, with an order of 6291456 and a nonzero element count of 52133888. The matrices can be downloaded from https://www.solver-conference.cn/solverchallenge23/index.html. Due to the evaluation from the above experiments, it is believed that ReproBLAS is relatively good. It supports multi process parallelism, is relatively robust, and has relatively good performance. We present a comprehensive comparison of the three BLAS libraries, as shown in Table 17. The following shows the results of three iterative methods using ReproBLAS and Jacobi preconditioner.

The results in Tables 18, 19, and 20 were obtained using 24 processes. Using 48 processes would yield the same results as the tables mentioned above. Therefore, all three iterative methods have achieved reproducible solutions for solving the three-temperature conduction algebraic equations. However, due to the poor convergence and high cost of the GMRES iterative method for solving the three temperature heat conduction algebraic equations, only the results within 1000 iterations are shown in Table 19. For the three temperature heat conduction algebraic equations, the PCG method exhibits better convergence performance compared to the other two iterative methods.

Table 17: Summary comparison of three BLAS.

	OzBLAS	ExBLAS	ReproBLAS
Reproducibility	✓	✓	✓
Performance	✓✓	✓	✓✓✓
Accuracy	✓✓	✓✓	✓
Robustness	✓	✓	✓✓
Parallelism	✓	✓✓	✓✓

Table 18: Residuals of PCG and PBiCGSTAB.

Iterations	PCG	PBiCGSTAB
0	6.325269545856831e+12	6.325269545856831e+12
1000	7.514178385434491e+05	7.386817478006695e+04
5000	1.436165216558322e+05	1.372804363645939e+04
8000	7.542657453696156e+04	1.386631296210173e+04
9000	6.294571636763243e+04	6.415859545534139e+04
10000	6.656557468221060e+04	3.930421787983290e+04

Table 19: Residuals of PGMRES.

Iterations	PGMRES
0	6.924941e+11
100	1.048481e+11
500	7.009688e+10
800	6.306948e+10
900	6.159947e+10
1000	6.006112e+10

Table 20: Direct error of PCG and PBiCGSTAB.

Iterations	PCG	PBiCGSTAB
0	1.000000000000000e+00	1.000000000000000e+00
1000	1.705911371201518e-01	4.113566427006439e-01
5000	7.556297004482126e-02	1.253485274264645e-01
8000	4.789952340114190e-02	7.170239112193111e-02
9000	3.671358362506560e-02	2.679057705850555e-01
10000	3.164842179260152e-02	2.804192571055464e-01

6 Related Work

Regarding the reproducible Krylov subspace method. Iakymchuk et al. implemented a reproducible parallel preconditioned CG algorithm (Jacobi preconditioner) based on ExBLAS in a pure MPI environment [5]. In addition, they also implemented a reproducible CG method in the MPI+OpenMP environment [8]. Iakymchuk et al. proposed a reproducible parallel preconditioned BiCGSTAB (Jacobi preconditioner) based on ExBLAS [7]. Lei et al. proposed a reproducible parallel preconditioned BiCGSTAB (Jacobi preconditioner) based on ExBLAS and ReproBLAS, and compared their reproducibility, accuracy, and performance [9]. Mukunoki et al. implemented the reproducibility of the CG solver on CPU and GPU [11].

Currently, the existing reproducible Krylov subspace methods are primarily based on BLAS implementations such as ExBLAS and ReproBLAS. Our work adds a comparison and analysis of OzBLAS. Most of the existing work on Krylov subspace methods has focused on the CG method and the BiCGSTAB method, while our work includes a comparison and analysis of the GMRES method. These three methods are the main Krylov subspace methods used, especially GMRES, which is very common in practical applications. Our work provides a comprehensive comparison of the three BLAS implementations and the reproducible parallel preconditioned Krylov subspace methods based on these BLAS. We also validated our findings on large-scale matrices from practical application problems.

7 Conclusions and Future Work

We highlighted the challenge of reproducibility in parallel preconditioned Krylov subspace algorithms. Initially, we examined the causes behind the irreproducibility of parallel iterative methods, pinpointing distributed dot products as the primary factors contributing to non-deterministic behavior. For achieving reproducible distributed dot products, we employed three methods: OzBLAS, ExBLAS, and ReproBLAS. Numerical experiments showed that the iterative method based on OzBLAS achieved reproducibility in a multi-threaded environment, the number of iterations, residuals, and errors were identical bit by bit. While the iterative method based on ExBLAS and ReproBLAS achieved reproducibility in a multi process environment, the number of iterations, residuals, and errors were identical bit by bit. Comparing the average time of several BLAS, the order from fast to slow is ReproBLAS, Regular, OzBLAS, ExBLAS. Furthermore, we generated ill-conditioned dot product data to compare the precision of several BLAS implementations, ranked in descending order of precision as OzBLAS, ExBLAS, ReproBLAS, and Regular. The precision of OzBLAS and ExBLAS is nearly identical. Finally, we applied the constructed reproducible iterative method to solve large-scale linear equations discretized from the three-temperature heat conduction equation, achieving reproducible solutions for the three-temperature heat conduction linear algebraic equations.

In the future, we will increase the comparison of RARE-BLAS. We also intend to compare reproducible parallel preconditioned iterative method based on four BLAS implementations in a mixed MPI+OpenMP environment. Furthermore, we will investigate the reproducibility of more complex preconditioners such as AMG and ILU.

Acknowledgments

The second author was supported in part by NSF of China (No. 12271055) and the foundation of key laboratory of computational physics, China. The third author was supported by National Natural Science Foundation of China (No. 62032023). The fourth author was supported by the InterFLOP (ANR-20-CE46-0009) project of the French National Agency for Research (ANR).

References

[1] Peter Ahrens, James Demmel, and Hong Diep Nguyen. 2020. Algorithms for efficient reproducible floating point summation. *ACM Transactions on Mathematical Software (TOMS)* 46, 3 (2020), 1–49.

[2] Richard Barrett, Michael Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. 1994. *Templates for the solution of linear systems: building blocks for iterative methods.* SIAM.

[3] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1–25.

[4] James Demmel, Jason Riedy, and Peter Ahrens. 2018. Reproducible BLAS: Make addition associative again! *SIAM News* 51, 8 (2018), 8.

[5] Roman Iakymchuk, Maria Barreda, Matthias Wiesenberger, José I Aliaga, and Enrique S Quintana-Ortí. 2020. Reproducibility strategies for parallel preconditioned conjugate gradient. *J. Comput. Appl. Math.* 371 (2020), 112697.

[6] Roman Iakymchuk, Sylvain Collange, David Defour, and Stef Graillat. 2015. ExBLAS: Reproducible and accurate BLAS library. In *NRE: Numerical Reproducibility at Exascale.*

[7] Roman Iakymchuk, Stef Graillat, and José I Aliaga. 2022. General framework for deriving reproducible Krylov subspace algorithms: BiCGStab case. In *International Conference on Parallel Processing and Applied Mathematics.* Springer, 16–29.

[8] Roman Iakymchuk, Maria Barreda Vayá, Stef Graillat, José I Aliaga, and Enrique S Quintana-Orti. 2020. Reproducibility of parallel preconditioned conjugate gradient in hybrid programming environments. *The International Journal of High Performance Computing Applications* 34, 5 (2020), 502–518.

[9] Xiaojun Lei, Tongxiang Gu, Stef Graillat, Xiaowen Xu, and Jing Meng. 2023. Comparison of reproducible parallel preconditioned BiCGSTAB algorithm based on ExBLAS and ReproBLAS. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region.* 46–54.

[10] Daichi Mukunoki, Takeshi Ogita, and Katsuhisa Ozaki. 2019. Reproducible BLAS routines with tunable accuracy using ozaki scheme for many-core architectures. In *International Conference on Parallel Processing and Applied Mathematics.* Springer, 516–527.

[11] Daichi Mukunoki, Katsuhisa Ozaki, Takeshi Ogita, and Roman Iakymchuk. 2021. Conjugate Gradient Solvers with High Accuracy and Bit-wise Reproducibility between CPU and GPU using Ozaki scheme. In *The International Conference on High Performance Computing in Asia-Pacific Region.* 100–109.

[12] Katsuhisa Ozaki, Takeshi Ogita, Shin'ichi Oishi, and Siegfried M Rump. 2012. Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. *Numerical Algorithms* 59 (2012), 95–118.

[13] Siegfried M Rump, Takeshi Ogita, and Shin'ichi Oishi. 2009. Accurate floating-point summation part II: Sign, K-fold faithful and rounding to nearest. *SIAM Journal on Scientific Computing* 31, 2 (2009), 1269–1302.

[14] Yousef Saad. 2003. *Iterative methods for sparse linear systems.* SIAM.

[15] Oreste Villa, Daniel Chavarria-Miranda, Vidhya Gurumoorthi, Andrés Márquez, and Sriram Krishnamoorthy. 2009. Effects of floating-point non-associativity on numerical computations on massively multithreaded systems. In *Proceedings of Cray User Group Meeting (CUG)*, Vol. 3.

[16] Naoya Yamanaka, Takeshi Ogita, Siegfried M Rump, and Shin¡¯ichi Oishi. 2008. A parallel algorithm for accurate dot product. *Parallel Comput.* 34, 6-8 (2008), 392–410.