# Comparison of Reproducible Parallel Preconditioned BiCGSTAB Algorithm Based on ExBLAS and ReproBLAS

Xiaojun Lei, Tongxiang Gu, Stef Graillat, Xiaowen Xu, Jing Meng

# Comparison of Reproducible Parallel Preconditioned BiCGSTAB Algorithm Based on ExBLAS and ReproBLAS

Xiaojun Lei[1], Tongxiang Gu[2], Stef Graillat[3(✉)], Xiaowen Xu[2,4], and Jing Meng[5]

[1] Graduate School of Chinese Academy of Engineering Physics, Beijing 100088, China
leixiaojun19@gscaep.ac.cn
[2] Institute of Applied Physics and Computational Mathematics, Beijing 100094, China
txgu@iapcm.ac.cn
[3] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
stef.graillat@lip6.fr
[4] CAEP Software Center for Numerical Simulation, Beijing 100094, China
xwxu@iapcm.ac.cn
[5] School of Mathematics and Statistics, Taishan University, Taian, 271000, China
mengpersist56@163.com

**Abstract.** Krylov subspace algorithms are important methods for solving linear systems. In order to solve large-scale linear systems and speed up the solution of linear systems, one has to use parallelism techniques. However, parallelism often enlarge the non-associativity of floating-point operations. This can lead to non-reproducibility of the computations. This paper compares the performance of the parallel preconditioned BiCGSTAB algorithm implemented with two different libraries (ExBLAS and ReproBLAS) that can ensure reproducibility of the computations. To address the effect of the compiler, we explicitly utilize the fma instructions. Finally, numerical experiments show that the BiCGSTAB algorithms based on the two BLAS implementations are reproducible, the BiCGSTAB algorithm based on ExBLAS is more accurate but more time-consuming, and the BiCGSTAB algorithm based on ReproBLAS is relatively less accurate but less expensive.

**Keywords:** floating-point arithmetic · reproducibility · ExBLAS · ReproBLAS · Parallel Preconditioned BiCGSTAB.

## 1 Introduction and Motivation

In many scientific and engineering calculation fields, such as nuclear reactor simulation, radiation (magneto) hydrodynamics, radiation diffusion problems, oil and gas resource exploration, numerical weather prediction, etc. [26–28], differential equations are often used as mathematical models to describe problems. In order to simulate on a computer, it is necessary to discretize the differential equations

to obtain a set of linear equations. Therefore, efficiently solving linear equations is the key to numerical simulation. For large-scale sparse matrices generated in actual problems, iterative methods based on Krylov subspace [8] are more advantageous because of the high computational complexity of direct solvers. However, the direct use of iterative methods may not converge or convergence can be very slow. The usual remedy is to use preprocessing techniques.

When the matrix is symmetric and positive definite, the Conjugate Gradient (CG) method [21] is one of the most effective Krylov subspace methods. When the matrix is unsymmetric, the BiConjugate Gradient Stabilized (BiCGSTAB) [22] and Generalized Minimal Residual (GMRES) methods [23] are the preferred Krylov subspace methods. BiCGSTAB performs better than GMRES for problems with complex eigenvalues [24]. Therefore, the BiCGSTAB method being a commonly used Krylov subspace iteration method. Getting bitwise identical floating-point results from multiple runs of the same program is a property that many users need for example for debugging or correctness checking in many codes [30]. Reproducibility is also needed for verification and validation, and even contractual obligations [25]. For the above reasons, we may expect that the results of the sequential and parallel implementations of Preconditioned BiCGSTAB (abbreviated as PBiCGSTAB) are identical, for instance, in the number of iterations, the intermediate and final residuals, as well as the output vector. However, in practice this is not often the case due to different reduction trees - the Message Passing Interface (MPI) implementations (libraries) [9] offer up to 14 different implementations for reduction -, data alignment, instructions used, order of the input data, etc. Each of these factors impacts the order of floating-point operations especially additions that are commutative but not associative. This can lead to non-reproducible results.

When parallel Preconditioned BiCGSTAB is used to solve linear equations, different numbers of processes will get different results, and the results of parallel Preconditioned BiCGSTAB program and serial Preconditioned BiCGSTAB program will also be different. This brings difficulties to verification and debugging. Therefore, our aim is to implement reproducible parallel Preconditioned BiCGSTAB algorithm that can achieve the same result using different number of processes on the same platform. For that, we implemented the reproducible parallel precondition BiCGSTAB solver using the two main BLAS libraries that can ensure reproducibility for basic operations; these are the ExBLAS and the ReproBLAS libraries, respectively. The ExBLAS library is more accurate, but more time-consuming, and the ReproBLAS library is less computationally expensive.

The rest of the paper is organized as follows. Section 2 introduces the ExBLAS method and the ReproBLAS method. Section 3 introduces the PBiCGSTAB algorithms and describes in detail its MPI implementation. We present strategies for ensuring reproducibility of PBiCGSTAB in Section 4 and evaluate corresponding implementations in Section 5. Section 6 introduces some related work. Finally, Section 7 draws conclusions and proposes future work.

## 2 Methodology

### 2.1 ExBLAS

The ExBLAS project [4] is an attempt to derive fast, accurate, and reproducible BLAS library by constructing a multi-level approach for these operations that are tailored for various modern architectures with their complex multilevel memory structures. ExBLAS combines together long accumulator and floating-point expansion (FPE) into algorithmic solutions. It has two features. On one side, this method aims to save each bit of information before finally rounding to the desired format. It is accurate and correctly rounded , so it is reproducible. On the other side, it is effectively tuned and implemented on various architectures, including conventional CPUs, Nvidia and AMD GPUs, and Intel Xeon Phi co-processors.

The corner stone of ExBLAS is reproducible parallel reduction. The ExBLAS parallel reduction relies upon FPEs with the `TwoSum` error-free transformation (EFT) [29] and long accumulators. In practice, the latter is invoked only once per overall summation that results in the little overhead (less than 8%) on accumulating large vectors. For details of the algorithm, please refer to [20]. In this article, we are concerned with the distributed dot product of two vectors. The dot product problem can be transformed into a summation problem using the `TwoProd` EFT [1]. The ExBLAS library also contains other routines, such as matrix vector multiplication, triangular solver and matrix matrix multiplication. The library is available at https://github.com/riakymch/exblas.

### 2.2 ReproBLAS

ReproBLAS aims at providing users with a set of (Parallel and Sequential) Basic Linear Algebra Subprograms that guarantee reproducibility regardless of the number of processors, of the data partitioning, of the way reductions are scheduled, and more generally of the order in which the sums are computed. ReproBLAS has three assumptions: (1) Floating-point numbers are binary and correspond to the IEEE 754-2008 Floating-Point Standard. (2) Floating-point operations are performed in round-to-nearest mode (ties may be broken arbitrarily). (3) Underflow occurs gradually (subnormal numbers must be supported) [25].

The corner stone of ReproBLAS [7] is reproducible summation, which is independent of the summation order. The reproducible summation algorithm can be found in [13, 14]. It is communication-optimal, in the sense that it does just one pass over the data in the sequential case, or one reduction operation in the parallel case, requiring an "accumulator" represented by just 6 floating-point words (more can be used if higher precision is desired). The arithmetic cost with a 6-word accumulator is $7n$ floating-point additions to sum $n$ words, and (in IEEE double precision) the final error bound can be up to $10^{-8}$ times smaller than the error bound for conventional summation. Let us denote $T = \sum_{j=0}^{n-1} x_j$, and $\overline{T}$ the floating-point approximation ot the summatino obtained with ReproBLAS. The absolute error of the reproducible summation algorithm

is $|T - \overline{T}| \leq n2^{W(1-K)} \max |x_j| + 7u| \sum_{j=0}^{n-1} x_j|$ where $u$ is the unit roundoff. The default configuration under double precision is $W = 40, K = 3$ [7]. Similarly, the dot product problem is transformed into a summation problem.

## 3 Algorithm(s)

We first state the Preconditioned BiCGSTAB algorithm, and then state the parallel Preconditioned BiCGSTAB algorithm.

### 3.1 Preconditioned BiCGSTAB

The BiCGSTAB algorithm was developed to solve nonsymmetric linear systems. The algorithmic description of the classical iterative PBiCGSTAB is presented in Algorithm 1 [3]. The loop body consists of sparse matrix-vector products (SpMV), dot products, AXPY (-like) operations, the preconditioner application, and a few scalar operations. For simplicity, in our implementation of the PBiCGSTAB method, we integrate the Jacobi preconditioner, which is composed of the diagonal elements of the matrix. As a consequence, the use of Jacobi preconditioner requires two vectors to be multiplied element by element.

### 3.2 Message-Passing Parallel PBiCGSTAB

In our parallel Preconditioned BiCGSTAB algorithm, we use $M$ processes. The matrix $A$ is partitioned into $M$ blocks of rows $(A_1, A_2, \ldots, A_i, \ldots, A_M)$, where each process stores one row-block. Vectors are partitioned and distributed in the same way as with the block-row distribution of $A$. In lines 13 and 17 of Algorithm 1, both need to calculate a SpMV, we call $MPI\_Allgatherv()$ to synthesize the local vector of each process into a complete vector. At lines 4, 14, 15, 18, and 21 of Algorithm 1, the distributed dot product needs to be computed, for non-reproducible parallel Preconditioned BiCGSTAB, we call $MPI\_Allreduce$. We have implemented a reproducible distributed dot product, which is different from the ordinary dot product. This is also the main difference between the reproducible parallel Preconditioned BiCGSTAB and the non-reproducible parallel Preconditioned BiCGSTAB. For details on reproducible distributed dot products, please see Section 4.

## 4 Strategies for Reproducibility

In [18], the article implements a reproducible parallel Preconditioned BiCGSTAB algorithm based on ExBLAS, also implemented with its lighter FPE-based version. In this paper, we implement a reproducible parallel Preconditioned BiCGSTAB algorithm based on ReproBLAS, and compare the performance of the solvers based on ExBLAS and ReproBLAS. In this section, we state our reproducible

**Algorithm 1:** The Preconditioned BiConjugate Gradient Stabilized Method `(PBiCGSTAB)` [3].

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
Choose $\tilde{r}$ (for example, $\tilde{r} = r^{(0)}$)
`for` $i = 1, 2, \ldots$
    $\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
    `if` $\rho_{i-1} = 0$ `method fails`
    `if` $i = 1$
        $p^{(i)} = r^{(i-1)}$
    `else`
        $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$
        $p^{(i)} = r^{(i-1)} + \beta_{i-1}(p^{(i-1)} - \omega_{i-1} v^{(i-1)})$
    `endif`
    `solve` $M\hat{p} = p^{(i)}$
    $v^{(i)} = A\hat{p}$
    $\alpha_i = \rho_{i-1}/\tilde{r}^T v^{(i)}$
    check norm of $s$; if small enough: set $x^{(i)} = v^{(i-1)} + \alpha_i \hat{p}$ and stop
    `solve` $M\hat{s} = s$
    $t = A\hat{s}$
    $\omega^{(i)} = t^T s / t^T t$
    $x^{(i)} = x^{(i-1)} + \alpha_i \hat{p} + \omega^{(i)} \hat{s}$
    $r^{(i)} = s - \omega^{(i)} t$
    check convergence; continue if necessary
    for continuation it is necessary that $\omega^{(i)} \neq 0$
`end`

```
double_binned *isum = NULL;
double_binned *local_isum = binned_dballoc(K);
binnedBLAS_dbddot(K, n, r, 1, r, 1, local_isum);
MPI_Reduce(local_isum, isum, 1, binnedMPI_DOUBLE_BINNED(K),
binnedMPI_DBDBADD(K), 0, MPI_COMM_WORLD);
if(myId == 0){
    tol = binned_ddbconv(K, isum);
}
MPI_Bcast(&tol, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

**Listing 1.** Reproducible Allreduce with ReproBLAS.

strategy for the parallel Preconditioned BiCGSTAB algorithm. Let us now inspect the sources of non-deterministic computations in the PBiCGSTAB solver as well as present our mitigation strategies for them.

*Dot products:* The main issue of non-determinism emerges from dot products and, thus, parallel reductions such as $MPI\_Allreduce()$. We use the ExBLAS method to provide a reproducible and correctly rounded dot product, and use the ReproBLAS method to provide a reproducible dot product. In [6], the authors provided a pseudocode for implementing distributed dot product with ExBLAS. Listing 1 provides pseudocodes for our implementation of the distributed dot product using the ReproBLAS.

*Sparse matrix-vector product:* Each process has a local $A_i$, and the complete vector is obtained by using $MPI\_Allgatherv()$, then each process calculates a SpMV, since the computations are carried locally and sequentially, they are deterministic and, thus, reproducible.

*AXPY(-type) vector updates:* For computing the axpy(-type) vector updates, we rely upon the sequential MKL implementation of axpy(-type).

*Application of the preconditioner:* We are using Jacobi preconditioner. The application of the Jacobi preconditioner is rather simple: first, the inverse of the diagonals is computed and then the application of the preconditioner only involves element-wise multiplication of two vectors. Thus, this part is both correctly rounded and reproducible.

## 5 Numerical Results

### 5.1 Setup

The following experiment is performed on Sugon HPC cluster with 172 compute nodes (16 GPU nodes), consisting of two 12-core Intel E5-2680 v3 processors each (24 cores per node). Each GPU computing node is equipped with 1 NVIDIA TESLA K80 GPGPU accelerator card (2.911TFlops). Nodes are connected by Intel Omni-Path high-speed computing network. The peak performance of the whole system is 211.2TFlop/s, and the memory is 64GB. The OS used by the cluster is Redhat7.2. The MPI library used for this experiment is MPICH.

For the experimental analysis, we leveraged a sparse (symmetric positive definite) coefficient matrix arising from the finite-difference method of a 3D Poisson's equation with 27 stencil points. In the following experiment, we use the ill-conditioned matrix in [6]. This generator scales the first row and the first column of the matrix so that dot product attains specified condition number and, hence, the matrix.

The right-hand side vector $b$ in the iterative solvers was always initialized to the product $A(1, 1, \ldots, 1)^T$, and the PBiCGSTAB iteration was started with the initial guess $x_0 = 0$. The parameter that controls the convergence of the iterative process is $\|r^j\|_2 \leq 10^{-8}$, where $j$ is the number of iterations.

### 5.2 Accuracy and Reproducibility Evaluation

In this section, we report the results of the accuracy and reproducibility evaluation. Additionally, we derive a sequential version of the code that relies on the GNU Multiple Precision Floating-Point Reliably (MPFR) library [5] - a C library for multiple (arbitrary) precision floating-point computations on CPUs - as a highly accurate reference implementation. This implementation uses 2048 bits of accuracy for computing dot product (192 bits for internal product of two floating-point numbers) and performs correct rounding of the computed result to double precision.

We analyzed the reproducibility of the reproducible parallel PBiCGSTAB. Hereafter, we will call them ExBLAS and ReproBLAS for short. In ReproBLAS, there is a parameter $K$ controlling accuracy, when there is no special description, we take the default $K = 3$. With double as the working precision, when $W = 40, K = 3$, $|T - \overline{T}| \leq n2^{40\times(-2)} \max|x_j| + 7 \times 2^{-53}|\sum_{j=0}^{n-1} x_j|$. When $K \geq 3$, $|T - \overline{T}| \approx 7 \times 2^{-53}|\sum_{j=0}^{n-1} x_j|$, therefore choose the smallest $K$ that can achieve the highest summation accuracy. In Table 1, the matrix size is $n = 4,019,679$ and the condition number is $10^{12}$. Table 1 shows the 2-norm of the intermediate and final residual of the PBiCGSTAB solver in each iteration, i.e. $\|r^j\|_2$. Table 2 shows the 2-norm of the intermediate and final residual of the ReproBLAS in each iteration when $K = 2$ and $K = 3$. We used one node on the Sugon cluster with 24 processes each pinned to one core. We present only few iterations, however the difference is present on all iterations. The ExBLAS implementation delivers both accurate and reproducible results that are identical with the MPFR library. The ReproBLAS implementation delivers reproducible results. Furthermore, we have also computed the direct error (see Table 3). Table 3 shows the infinite norm of the approximate and exact solutions of the PBiCGSTAB solver in each iteration, i.e. $\|x^j - x^*\|_\infty$. Note that the result on ExBLAS is identical to MPFR. Table 4 shows the infinite norm of the approximate and exact solutions of the ReproBLAS in each iteration when $K = 2$ and $K = 3$.

### 5.3 Performance Evaluation

We first analyze the performance of regular dot product, ExBLAS-based and Reproblas-based dot product. Hereafter, we will call them ExBLASdot and

**Table 1.** Accuracy and reproducibility comparison on the intermediate and final residual against MPFR for a matrix with condition number of $10^{12}$. The matrix is generated following the procedure from Section 5.1 with $n = 4,019,679(159^3)$.

| Residual | | | | | |
|---|---|---|---|---|---|
| Iteration | MPFR | Original 1 proc | Original 48 proc | ExBLAS 48 proc | ReproBLAS 48 proc($K = 3$) |
| 0 | 6.199998000000062e+14 | 6.199998000000064e+14 | 6.199998000000064e+14 | 6.199998000000062e+14 | 6.199998000000062e+14 |
| 2 | 7.773390285775344e+07 | 7.773396235537358e+07 | 7.773396235219534e+07 | 7.773390285775344e+07 | 7.773396235223217e+07 |
| 18 | 1.800391538018418e+02 | 9.920664169207433e+02 | 7.848983492248201e+02 | 1.800391538018418e+02 | 3.647699220519532e+01 |
| 19 | 4.527368042985061e+01 | 9.920511721210302e+02 | 1.708520638885489e+02 | 4.527368042985061e+01 | 1.377536701580542e+01 |
| ... | ... | ... | ... | ... | ... |
| 38 | 1.572043303241533e-06 | 3.155149206702169e-03 | 2.941521670086655e-04 | 1.572043303241533e-06 | 2.215388966485101e-07 |
| 39 | 2.453500079721513e-07 | 1.269826897510669e-03 | 2.783864941272381e-04 | 2.453500079721513e-07 | 8.834134369067694e-08 |
| 40 | 3.854451111920641e-08 | 1.237807074881981e-03 | 4.132770031308131e-05 | 3.854451111920641e-08 | 6.701333941850470e-09 |
| 41 | 6.068369560766913e-09 | 4.530740507879870e-05 | 4.060713358832429e-05 | 6.068369560766913e-09 | |
| 49 | | 7.217500962630385e-08 | 2.373922675763727e-08 | | |
| 50 | | 1.275390456398120e-08 | 3.914596612172984e-09 | | |
| 51 | | 9.575558932042771e-09 | | | |

**Table 2.** Comparison of the reproducibility of the intermediate and final residuals of the matrix with a condition number of $10^{12}$. The matrix is generated following the procedure from Section 5.1 with $n = 4,019,679(159^3)$.

| Residual | | | | |
|---|---|---|---|---|
| Iteration | ReproBLAS 24 proc($K = 2$) | ReproBLAS 48 proc($K = 2$) | ReproBLAS 24 proc($K = 3$) | ReproBLAS 48 proc($K = 3$) |
| 0 | 6.199998000000062e+14 | 6.199998000000062e+14 | 6.199998000000062e+14 | 6.199998000000062e+14 |
| 2 | 7.773396235223238e+07 | 7.773396235223238e+07 | 7.773396235223217e+07 | 7.773396235223217e+07 |
| 18 | 5.598082147961621e+02 | 5.598082147961621e+02 | 3.647699220519532e+01 | 3.647699220519532e+01 |
| 19 | 1.401936884527657e+02 | 1.401936884527657e+02 | 1.377536701580542e+01 | 1.377536701580542e+01 |
| ... | ... | ... | ... | ... |
| 38 | 6.514290840710756e-05 | 6.514290840710756e-05 | 2.215388966485101e-07 | 2.215388966485101e-07 |
| 39 | 1.434720487921011e-05 | 1.434720487921011e-05 | 3.834134369067694e-08 | 3.834134369067694e-08 |
| 40 | 1.187160733888966e-05 | 1.187160733888966e-05 | 6.701333941850470e-09 | 6.701333941850470e-09 |
| 41 | 2.536931803781759e-06 | 2.536931803781759e-06 | | |
| 42 | 2.193618822360498e-06 | 2.193618822360498e-06 | | |
| 43 | 2.815475613230721e-07 | 2.815475613230721e-07 | | |
| 44 | 3.861719136834829e-08 | 3.861719136834829e-08 | | |
| 45 | 7.197952430468738e-09 | 7.197952430468738e-09 | | |

**Table 3.** Direct error comparison against MPFR for a matrix with condition number of $10^{12}$. The matrix is generated following the procedure from Section 5.1 with $n = 4,019,679(159^3)$.

| Direct error | | | | | |
|---|---|---|---|---|---|
| Iteration | MPFR | Original 1 proc | Origina 48 proc | ExBLAS 48 proc | ReproBLAS 48 proc($K = 3$) |
| 0 | 1.000000000000000e+00 | 1.000000000000000e+00 | 1.000000000000000e+00 | 1.000000000000000e+00 | 1.000000000000000e+00 |
| 2 | 9.468739489168004e+04 | 9.468739489167897e+04 | 9.468739489167897e+04 | 9.468739489168004e+04 | 9.468739489167897e+04 |
| 18 | 3.759048477414416e-01 | 2.473965872164404e+00 | 4.032131048428883e-01 | 3.759048477414416e-01 | 5.105001804315423e-02 |
| 19 | 5.816123121107641e-02 | 3.747532604204130e-01 | 4.049929311272970e-01 | 5.816123121107641e-02 | 9.454486314514732e-03 |
| ... | ... | ... | ... | ... | ... |
| 38 | 8.071978641055466e-10 | 4.484255959891215e-07 | 1.594126798343254e-07 | 8.071978641055466e-10 | 1.228213974968639e-09 |
| 39 | 8.072296164840509e-10 | 4.481986183302311e-07 | 3.417345129097527e-08 | 8.072296164840509e-10 | 1.228311674594806e-09 |
| 40 | 8.076090907138678e-10 | 1.625782982683788e-07 | 3.417765470636880e-08 | 8.076090907138678e-10 | 1.228331658609250e-09 |
| 41 | 8.076523894118282e-10 | 1.630189810919447e-07 | 3.606841825209983e-09 | 8.076523894118282e-10 | |
| 49 | | 7.744984653612619e-10 | 8.554250641168437e-10 | | |
| 50 | | 7.744849206403615e-10 | 8.554250641168437e-10 | | |
| 51 | | 7.745686314564182e-10 | | | |

**Table 4.** Direct error comparison of matrix with condition number $10^{12}$. The matrix is generated following the procedure from Section 5.1 with $n = 4,019,679(159^3)$.

| Residual | | | | |
|---|---|---|---|---|
| Iteration | ReproBLAS 24 proc($K = 2$) | ReproBLAS 48 proc($K = 2$) | ReproBLAS 24 proc($K = 3$) | ReproBLAS 48 proc($K = 3$) |
| 0 | 1.000000000000000e+00 | 1.000000000000000e+00 | 1.000000000000000e+00 | 1.000000000000000e+00 |
| 2 | 9.468739489167897e+04 | 9.468739489167897e+04 | 9.468739489167897e+04 | 9.468739489167897e+04 |
| 18 | 2.044747069809166e-01 | 2.044747069809166e-01 | 5.105001804315423e-02 | 5.105001804315423e-02 |
| 19 | 2.044624037738916e-01 | 2.044624037738916e-01 | 9.454486314514732e-03 | 9.454486314514732e-03 |
| ... | ... | ... | ... | ... |
| 38 | 3.604734521989172e-08 | 3.604734521989172e-08 | 1.228213974968639e-09 | 1.228213974968639e-09 |
| 39 | 6.525545215296802e-09 | 6.525545215296802e-09 | 1.228311674594806e-09 | 1.228311674594806e-09 |
| 40 | 1.458647203023133e-09 | 1.458647203023133e-09 | 1.228331658609250e-09 | 1.228331658609250e-09 |
| 41 | 1.229433221894283e-09 | 1.229433221894283e-09 | | |
| 42 | 1.230030521881531e-09 | 1.230030521881531e-09 | | |
| 43 | 1.230010315822483e-09 | 1.230010315822483e-09 | | |
| 44 | 1.230061830170825e-09 | 1.230061830170825e-09 | | |
| 45 | 1.230070045821208e-09 | 1.230070045821208e-09 | | |

ReproBLASdot for short. We use the same data generation method as the ReproBLAS library [7], i.e. $a_i = \sin(2.0 * \pi * (i/n - 0.5))$, $b_i = \sin(2.0 * \pi * (i/n - 0.5))$, $n = 16200000$. Table 5 reports the execution time ratio of different dot products (averaged for 5 different executions). Figure 1 (a) reports the total execution time (averaged for 5 different executions) of the reproducible dot product for this cluster normalized with respect to the execution time of the regular MPI version, when we vary the number of cores. From table 5 and figure 1 (a), we can see that ReproBLASdot is faster than ExBLASdot product.

Then, we analyzed the performance of the reproducible parallel PBiCGSTAB. Our experiments evaluate the strong scaling of this reproducible implementation compared against the regular (non-reproducible) version of PBiCGSTAB using MPI. We analyze the performance of the three implementations in the aforementioned cluster. Specifically, in order to assess the strong scalability, we fix the matrix size to $n = 16,003,008$ and increase the number of cores. Table 6 reports the execution time ratio (averaged for 5 different executions) of the different MPI PBiCGSTAB solvers on this platform, varying the number of cores (from 48 to 144 in Sugon) as we maintain the problem size, the number of iterations in parentheses. Figure 1 (b) reports the total execution time (averaged for 5 different executions) of the reproducible MPI PBiCGSTAB solvers for this cluster normalized with respect to the execution time of the regular MPI version, when we vary the number of cores. From table 6 and figure 1 (b), we can see that ReproBLAS is faster than ExBLAS. We also found that ReproBLAS is faster than regular MPI version when using 72, 96, and 144 processes, because ReproBLAS has fewer iterations, so the total time is shorter when the same accuracy conditions are met. It can be seen from this example that although the dot product uses a reproducible dot product, each iteration pays a certain amount of overhead, but it may reduce the total number of iterations and reduce the total overhead. Table 7 reports the average time ratio of each iteration (averaged for 5 different executions) of the different MPI PBiCGSTAB solvers on this platform, varying the number of cores (from 48 to 144 in Sugon) as we maintain the problem size. Figure 2 (a) reports the average time of each iteration (averaged for 5 different executions) of the reproducible MPI PBiCGSTAB solvers for this cluster normalized with respect to the execution time of the regular MPI version, when we vary the number of cores. We fixed the total number of iterations to 60, figure 2 (b) reports the total execution time (averaged for 5 different executions) of the reproducible MPI PBiCGSTAB solvers for this cluster normalized with respect to the execution time of the regular MPI version.

From our experiment, we can observe the following facts:

- The dot product based on ReproBLAS is faster than that based on ExBLAS.
- Reproducible parallel PBiCGSTAB based on ExBLAS library can get the same results as MPFR. It is reproducible and accurate.
- Reproducible parallel PBiCGSTAB based on ReproBLAS library has the same result no matter how many processes are used, but it cannot get the same result as MPFR.

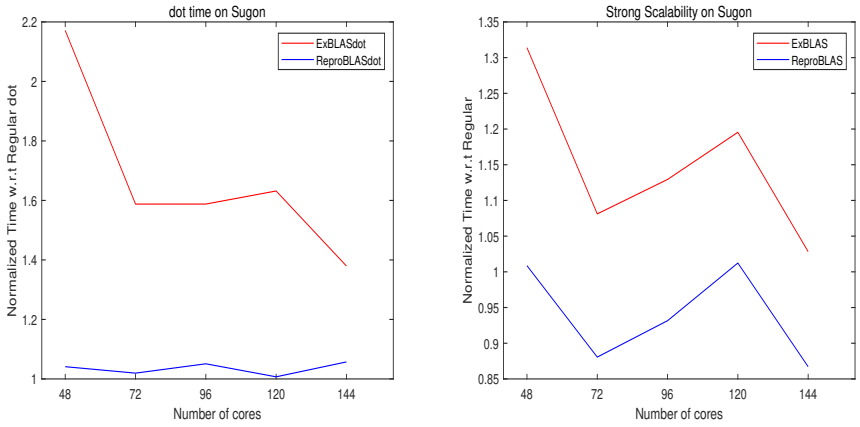


**Fig. 1.** dot product time (left) strong scalability (right)

**Table 5.** Different MPI implementations of dot product on Sugon, where one process is pinned to one core.

| Execution time ratios | | | |
|---|---|---|---|
| Number of cores | Regular dot | ExBLASdot | ReproBLASdot |
| 48 | 1 | 2.171 | 1.041 |
| 72 | 1 | 1.588 | 1.020 |
| 96 | 1 | 1.588 | 1.051 |
| 120 | 1 | 1.632 | 1.007 |
| 144 | 1 | 1.380 | 1.057 |

**Table 6.** Strong scalability of different MPI implementations of the PBiCGSTAB method on Sugon, where one process is pinned to one core.

| Execution time ratios | | | |
|---|---|---|---|
| Number of cores | Regular | ExBLAS | ReproBLAS |
| 48 | 1(43) | 1.314(43) | 1.009(43) |
| 72 | 1(50) | 1.081(43) | 0.881(43) |
| 96 | 1(47) | 1.129(43) | 0.932(43) |
| 120 | 1(43) | 1.195(43) | 1.012(43) |
| 144 | 1(50) | 1.028(43) | 0.867(43) |

**Table 7.** Time for one iteration of different MPI implementations of the PBiCGSTAB method on Sugon, where one process is pinned to one core.

| Execution time ratios | | | |
|---|---|---|---|
| Number of cores | Regular | ExBLAS | ReproBLAS |
| 48 | 1 | 1.314 | 1.009 |
| 72 | 1 | 1.257 | 1.024 |
| 96 | 1 | 1.234 | 1.018 |
| 120 | 1 | 1.195 | 1.012 |
| 144 | 1 | 1.197 | 1.009 |

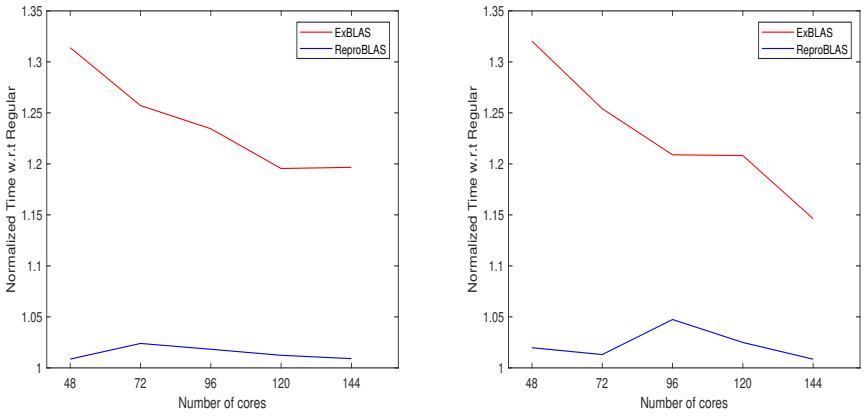


**Fig. 2.** One iteration time ratio (left) and total time ratio (right)

- When $K = 2$, the reproducible parallel PBiCGSTAB based on the ReproBLAS library is also reproducible, but the result is different from $K$ greater than or equal to 3.
- For ill-conditioned matrices, using a high-precision library to calculate the inner product may reduce the number of iterations. In our experiment, when the condition number is $10^4$, there is only one difference in the number of iterations. When the condition number is $10^{12}$, there is a difference of 10 in the number of iterations.
- The reproducible parallel PBiCGSTAB based on the ReproBLAS library is faster than the reproducible parallel PBiCGSTAB based on the ExBLAS library.

## 6 Related Work

Accuracy and reproducibility issues have different motivations and natures [10], but they are due to the same reason, which is caused by rounding errors. Although achieving reproducibility will not improve accuracy, increasing accuracy will help reduce the severity of reproducible problems. Here, we briefly introduce the reproducible work of hardware manufacturers, several BLAS, and Krylov subspace methods.

Intel has introduced a version of their Math Kernel Library (MKL) that supports reproducibility under certain restrictive conditions [11]. NVIDIA's cuBLAS routines are, by default, reproducible under the same conditions [12].

Collange, Defour, Graillat and Iakymchuk proposed ExBLAS [4]. ExBLAS is based on combining long accumulators and floating-point expansions in conjunction with error-free transformations. Demmel and Nguyen proposed a series of reproducible summation algorithms [13, 14]. Ahrens, Nguyen, and Demmel

extended their concept to few other reproducible BLAS routines, distributed as the ReproBLAS library [7]. Mukunoki and Ogita presented their approach to implement reproducible BLAS, called OzBLAS [15], with tunable accuracy. Chohra introduced RARE-BLAS (Reproducible, Accurately Rounded and Efficient BLAS) that benefits from recent accurate and efficient summation algorithms [16, 17].

Regarding the reproducible Krylov subspace method, Iakymchuk et al. realized the reproducibility of pure MPI parallel Preconditioned CG on the CPU based on ExBLAS, use Jacobi preconditioner [6]. Further, they realized the reproducibility of the pure MPI parallel Preconditioned BiCGSTAB algorithm on the CPU based on ExBLAS, use Jacobi preconditioner [18]. Furthermore, they have also achieved reproducibility in the MPI+OpenMP environment [19]. Mukunoki et al. realizes the reproducibility of the CG solver on the CPU and GPU [10].

## 7 Conclusions and Future Work

We emphasized the reproducibility problem of the parallel Preconditioned BiCGSTAB algorithm. We at first analyzed the MPI implementation of the PBiCGSTAB method and identified two major sources of non-deterministic behavior, namely dot products and compiler optimization. The latter may change the order of operations or replace some of them in favor of the fused multiply-add (fma) operation. To tackle compiler interference in computations, we reconstruct computations as well as explicitly invoke fma instructions. For reproducible and distributed dot product, we use two methods, which are based on ExBLAS and based on ReproBLAS. Both the reproducible parallel preconditioned BiCGSTAB method realize the reproducibility of the number of iterations, intermediate and final residuals and the final output vector. The ExBLAS method is more accurate than the ReproBLAS method, but the ReproBLAS method is faster. Since the dot product accounts for a relatively low percentage of computation time in the parallel preconditioned BiCGSTAB, when we fix the total number of iterations to 60, the average time cost of reproducible parallel preconditioned BiCGSTAB method based on ExBLAS is 1.23 times that of the non-reproducible parallel preconditioned BiCGSTAB, the average time cost of reproducible parallel preconditioned BiCGSTAB method based on ReproBLAS is only 1.02 times that of the non-reproducible parallel preconditioned BiCGSTAB.

In the future, we will add a comparison with OzBLAS and RARE-BLAS. We also intend to compare reproducible parallel preconditioned BiCGSTAB method based on four BLAS implementations in a mixed MPI+OpenMP environment.

**Acknowledgements** The second author was supported in part by Science Challenge Project, China (TZ2016002), NSF of China (61472462, 11671049, 11601033, 62032023) and the foundation of key laboratory of computational physics, China. The third author was supported by the InterFLOP (ANR-20-CE46-0009) project of the French National Agency for Research (ANR). The

fourth author was supported by National Natural Science Foundation of China (No. 62032023). The fifth author was supported by the Project of Natural Science Foundation of Shandong Province (no.ZR2021MA092).

## References

1. Ogita, T., Rump, S., Oishi, S.: Accurate sum and dot product. SIAM J. Sci. Comput. 26(6), 1955-1988 (2005)
2. Higham, N.: Accuracy and Stability of Numerical Algorithms. 2nd edn. SIAM Publications, Philadelphia (2002)
3. Review, B.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM Publications, (1995)
4. Iakymchuk, R., Collange, S., Defour, D., Graillat, S.: ExBLAS: Reproducible and accurate BLAS library. In: Proceedings of the NRE2015 Workshop Held as Part of SC15. Austin, TX, USA, November 15-20, (2015)
5. The MPFR library (2004). http://www.mpfr.org/. Accessed 8 July 2016
6. Iakymchuk, R., Barreda, M., Wiesenberger, M., Aliaga, J. I., Quintana-Ortí, E. S.: Reproducibility strategies for parallel Preconditioned Conjugate Gradient. Journal of Computational and Applied Mathematics. 371, 0377-0427 (2020)
7. Ahrens, P., Demmel, J., Nguyen, H.: Algorithms for Efficient Reproducible Floating Point Summation. ACM Transactions on Mathematical Software. 46(3), 0098-3500 (2020). https://doi.org/10.1145/3389360
8. Saad, Y.: Iterative Methods for Sparse Linear Systems. 3nd edn. SIAM, Philadelphia, PA, USA, (2003)
9. Gropp, W., Hoefler, T., Thakur, R., Lusk, E.: Using Advanced MPI: Modern Features of the Message-Passing Interface. MIT Press, (2014)
10. Mukunoki, D., Ozaki, T., Ogita, T., Iakymchuk, R.: Conjugate Gradient Solvers with High Accuracy and Bit-wise Reproducibility between CPU and GPU using Ozaki scheme. The International Conference on High Performance Computing in Asia-Pacific Region. 100-109 (2021)
11. Intel, Intel Math Kernel Library. http://software.intel.com/en-us/intel-mkl. 2020
12. NVIDIA, NVIDIA cuBLAS. https://developer.nvidia.com/cublas. 2021
13. Demmel, J., Nguyen, H. D.: Fast reproducible floating-point summation. In: Proceedings of 21th IEEE Symposium on Computer Arithmetic, Austin, Texas, USA (2013). doi: 10.1109/ARITH.2013.9
14. Demmel, J., Nguyen, H. D.: Parallel reproducible summation. IEEE Trans. Comput. 64(7), 2060-2070 (2015). doi: 10.1109/TC.2014.2345391
15. Mukunoki, D., Ogita, T., Ozaki, K.: Accurate and reproducible BLAS routines with Ozaki scheme for many-core architectures. in: Proc. International Conference on Parallel Processing and Applied Mathematics, PPAM2019, 516-527 (2020)
16. Chohra, C., Langlois, P., Parello, D.: Efficiency of reproducible level 1 BLAS. SCAN: Scientific Computing, Computer Arithmetic, and Validated Numerics, 99-108 (2015)
17. Chohra, C., Langlois, P., Parello, D.: Reproducible, accurately rounded and efficient BLAS. in Euro-Par Parallel Processing Workshops, 609-620 (2016)
18. Iakymchuk, R., Graillat, S., Aliaga, J.: General framework for deriving reproducible Krylov subspace algorithms: A BiCGStab case study (2021). https://hal.sorbonne-universite.fr/hal-03382119/document

19. Iakymchuk, R., Vayá, M. B., Graillat, S., Aliaga, J. I., Quintana-Ortí, E. S.: Reproducibility of parallel preconditioned conjugate gradient in hybrid programming environments. The International Journal of High Performance Computing Applications, 34(5), 502-518 (2020)
20. Collange, S., Defour, D., Graillat, S., Iakymchuk, R.: Numerical reproducibility for the parallel reduction on multi- and many-core architectures. Parallel Comput. 49(C), 83-97 (2015). http://dx.doi.org/10.1016/j.parco.2015.09.001
21. Fletcher, R.: Conjugate gradient methods for indefinite systems, in: G. A. Watson (Ed.), Numerical Analysis. Springer Berlin Heidelberg, Berlin, Heidelberg, 11, 73-89 (1976)
22. van der Vorst, H. A.: Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. SIAM J. Sci. Stat. Comput. 13, 631-644 (1992)
23. Saad, Y., Schultz, M. H.: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. Siam Journal on Scientific and Statistical Computing. 7, 856-869 (1968)
24. Zhang, S. L.: GPBi-CG: Generalized Product-type Methods Based on Bi-CG for Solving Nonsymmetric Linear Systems. Siam Journal on Scientific Computing. 18(2), (1997)
25. Nguyen, H. D., Demmel, J., Ahrens, P.: ReproBLAS: Reproducible BLAS. http://bebop.cs.berkeley.edu/reproblas/. 2018
26. Xu Xiaowen, Mo Zeyao, An Hengbin: Algebraic two-level iterative method for 2-D 3-T radiation diffusion equations. Chinese J Comput Phys. 26(1), 1-8 (2009)(in Chinese)
27. Dogru, A. H., Fung, L. S. K., Middya, U., Al-Shaalan, T. M., Byer, T., Hoy, H., et al: New frontiers in large scale reservoir simulation. In SPE Reservoir Simulation Symposium. Society of Petroleum Engineers, (2011)
28. Kimura, R.: Numerical weather prediction. Journal of Wind Engineering and Industrial Aerodynamics. 90(12), 1403-1414 (2002). https://doi.org/10.1016/S0167-6105(02)00261-1
29. Knuth, D. E.: The Art of Computer Programming: Seminumerical Algorithms. volume 2. Addison-Wesley, (1969)
30. Villa, O., Chavarria-Miranda, D., Gurumoorthi, V., Marquez, A., Krishnamoorthy, S.: Effects of floating-point nonassociativity on numerical computations on massively multithreaded systems. in Cray User Group meeting, CUG, (2009)