预印本 XX(X):1–14
©作者 2023

并行 Krylov 求解器中确保数值可靠性的通用框架：以 BiCGStab 方法为例

Roman Iakymchuk[1,2], Jos´e I. Aliaga[3]

摘要
Krylov 子空间方法的并行实现通常有助于加快求解线性系统近似解的过程。然而，这种并行化与异步执行和乱序执行相结合，往往会加剧浮点运算中的非结合性影响。使用通信隐藏流水线算法来改进 Krylov 子空间方法的并行化时，这些问题甚至会进一步加剧。在实现中引入可复现性，可以通过获得更稳健、正确的解来避免这些问题。本文提出了一个通用框架，用于推导 Krylov 子空间方法的可复现且准确的变体。所提出的算法策略辅以可编程性建议，以确保执行具有确定性和准确性。本文以预条件 BiCGStab 方法及其流水线变体为例，展示该框架；后者实际上是 Krylov 子空间方法家族中的一种独特方法，可通过消息传递求解非对称线性系统。最后，我们在 SuiteSparse 矩阵集合中的一组矩阵以及一个三维泊松方程上，验证了 BiCGStab 两种可复现变体的数值行为。

关键词
数值可靠性、可复现性、精度、ExBLAS、PBiCGStab、流水线 PBiCGStab、高性能计算。

arXiv:2302.04180v1 [cs.MS] 8 Feb 2023

1 引言

求解大型稀疏线性方程组是许多科学应用中的常见问题，应用领域涵盖电路与器件仿真、量子物理、大规模特征值计算，以及各种涉及偏微分方程（PDE）离散化的应用（Barrett 等，1994）。在此类问题中，Krylov 子空间方法承担着标准线性代数求解器的角色（Saad，2003）。共轭梯度（CG）法可视为这类迭代求解器的先驱，其适用于对称正定（SPD）系统。此后，人们提出了其他 Krylov 子空间方法，以求解更一般的非对称不定线性系统，包括广义最小残差法（GMRES）（Saad 和 Schultz，1986）、双共轭梯度法（BiCG）（Fletcher，1976）、共轭梯度平方（CGS）法（Sonneveld，1989），以及广泛使用的双共轭梯度稳定法（BiCGStab）（Van der Vorst，1992）；与上述前两种方法相比，该方法的收敛过程更平滑。此外，在这些方法的实际实现中，通常会加入预条件处理，以加快方法收敛并改善其数值特性。

人们通常会认为，Krylov 子空间方法的顺序实现与并行实现应当得到相同的结果，例如迭代次数、中间残差和最终残差，以及所求的解向量。然而，实践中情况往往并非如此，这是由于归约树各不相同——消息传递接口（MPI）库提供了多达 14 种不同的归约实现——数据对齐方式、所使用的指令等因素所致。这些因素都会影响浮点运算的顺序，而浮点运算满足交换律但不满足结合律，因此会破坏可复现性。我们的目标是确保计算结果（包括残差/误差）相同且精确；在我们看来，这是确保迭代方法鲁棒性 和 正确性 的一种方式。在这种情况下，鲁棒性和正确性有三个目标：可复现性*，结果同时具备精度保证，并采用可持续（节能的）算法解决方案。

Krylov 子空间方法在大规模并行系统上的实现暴露了其可扩展性问题。主要原因是全局通信（尤其是归约）所需的同步会拖慢并行执行。最常见的解决方案是开发通信规避方法，并采用新的 MPI 函数来隐藏通信，将其执行与迭代方法的计算重叠起来。在 Cools 和 Vanroose（2017）中，作者提出了一个用于推导流水线*Krylov*子空间方法的通用框架，通过重新设计递推关系来简化并行化。同样，这些改动也会影响迭代方法的稳健性和正确性。

[1]于默奥大学，瑞典 [2]索邦大学，
法国[2]海梅一世大学，西班牙

通讯作者： Roman Iakymchuk，于默奥大学，MIT-Huset，901 87 于默奥，瑞典 电子邮箱：riakymch@cs.umu.se

*可复现性是指在不同并行环境中，针对相同数据进行多次执行时，能够获得逐位相同且准确的结果。

一般而言，Krylov 子空间方法由三个部分构成：稀疏矩阵-向量乘法 Ax(SPMV)、两个向量之间的点积（$x,\ y$），以及将向量乘以标量后再与另一个向量相加 $y := \alpha x + y$(AXPY)。若采用分块数据分布，则只有 AXPY 在本地执行；SPMV 需要收集完整的操作数向量，例如通过 MPI Allgatherv() 集合通信实现；点积则需要在 MPI 进程之间进行通信和计算，例如通过 MPI Allreduce() 集合通信实现。虽然 SPMV 的浮点运算量（flops）最大，但在大规模情况下，由于需要进行全局通信，点积会成为 Krylov 子空间方法中最耗时的部分。这也说明了采用 Krylov 子空间方法流水线版本的合理性。

本文旨在重新确保并行环境中 Krylov 子空间方法的可复现性。我们的贡献如下：

- 我们提出了一个用于推导可复现 *Krylov* 子空间方法的通用框架。我们采用自底向上方法，通过确保其各组成部分（包括全局通信）的可复现性，来保证 Krylov 子空间方法的可复现性。我们的可复现方案建立在 ExBLAS Col-langeand 等人（2015）的方法及其轻量化版本之上。
- 即使采用我们的可复现方案，我们仍特别强调谨慎安排计算的重要性，例如避免编译器可能将 $a*b+c$ 替换为融合乘加（fma）运算，或在数据初始化时推迟除法运算（即先除后用）。例如，我们提供了使用 fma 的定制 AXPY（类）运算，将每次 AXPY（类）运算中的舍入误差减少到一到两次。我们参考了 Goldberg Goldberg（1991）撰写的、已有 30 年历史但至今仍不过时的指南《每位计算机科学家都应了解的浮点运算》。
- 我们在预条件 BiCGStab（PBiCGStab）方法和流水线PBiCGStab方法上验证了所提出方法的适用性和性能。我们为每种方法推导出两个可复现变体，并在一组 SuiteSparse 矩阵和一个三维泊松方程上对其进行测试。

本文的结构如下。第 2 节介绍计算机算术的若干方面以及 ExBLAS 方法。第 3 节提出用于构建可复现 Krylov 子空间方法的通用框架。第 4 节介绍 PBiCGStab 方法和流水线 PBiCGStab 方法，并详细描述其 MPI 实现。随后，我们在第 5 节评估 PBiCGStab 和流水线 PBiCGStab 的两种可复现实现。最后，第 6 节回顾相关工作，第 7 节给出结论并展望未来方向。

2 背景

首先，我们简要介绍浮点运算：它通过具有有限固定精度表示的数来近似实数，这些数由有效数、指数和符号组成：

$$x = \pm \underbrace{x_0.x_1 \ldots x_{M-1}}_{mantissa} \times b^e,\ 0 \le x_i \le b-1,\ x_0 \neq 0,$$

其中，b 表示基（在我们的案例中为 2），M 表示精度，e 表示有界的指数（$e_{\min} \le e \le e_{\max}$）。

IEEE 754 标准 IEEE 计算机学会（2008）创建于 1985 年，并于 2008 年和 2019 年进行了修订。该标准通过严格规定浮点运算的属性，显著提高了数值计算的可靠性。如今，大多数处理器都采用了这一标准，从而大幅提升了数值应用的可移植性。该标准规定了浮点格式，这些格式通常与 二进制*16*位、二进制*32*位 和 二进制*64*位 等精度相关，见表 1。浮点表示法使数字能够覆盖很大的动态范围，其定义为集合中绝对值最大的数与绝对值最小的非零数之比。例如，二进制64位（双精度）能够表示从4.9×10^{-324} 到 1.8×10^{308}的正数，因此其动态范围为 3.7×10^{631}。

IEEE 754 标准要求基本算术运算（+、−、×、/、√、融合乘加）产生正确舍入结果。这意味着，这些运算仿佛先以无限精度计算出结果，再将其舍入到相应的浮点格式。正确舍入标准保证了唯一且定义明确的答案，从而确保单次运算的逐位可复现性；但仅有正确舍入并不是实现可复现性的必要条件。计算机算术领域日益关注可复现性，力求更加审慎地关注这一问题。这促使 2019 年修订版 754 标准纳入了加法和乘法的无误差变换（EFT）：将精确结果和误差分别作为结果返回，以确保浮点运算的数值可复现性。这些机制一旦在硬件中实现，将简化我们的可复现算法，例如 ExBLAS Collange 等（2015）、ReproBLAS Demmel 和 Nguyen（2015）以及 OzBLAS Mukunoki 等（2019）库中使用的算法，并提升其性能。

有两种方法可以在不产生舍入误差的情况下进行浮点数加法，或减少舍入误差的影响。其核心思想是在计算过程中同时跟踪结果和误差。第一种方法使用 EFT 计算结果和舍入误差，并将它们存储在浮点展开（FPE）中。它是一种未求值的 p 浮点数之和，其分量按大小排序，彼此重叠极小，从而覆盖整个指数范围。通常，FPE 依赖传统的 EFT：加法使用 Knuth（1969）提出的 twosum，乘法使用 Ogita 等人（2005）提出的 twoprod。这两种运算的代码分别见算法1和算法2。第二种方法将浮点数有限的指数范围映射到一个称为长（定点）累加器的长向量中，并将每一位存储其中。例如，Kulisch 和 Snyder（2011）提出，

表 1. 三种 IEEE 算术精度的参数。

Type	Size	有效数	指数	舍入单位	范围
half	16 位	11 位	5 位	$u = 2^{-11} \approx 4.88 \times 10^{-4}$	$\approx 10^{\pm 5}$
单精度	32 位	24 位	8 位	$u = 2^{-24} \approx 5.96 \times 10^{-8}$	$\approx 10^{\pm 38}$
双精度	64 位	53 位	11 位	$u = 2^{-53} \approx 1.11 \times 10^{-16}$	$\approx 10^{\pm 308}$

使用 4288 位长累加器，对由二进制64位数构成的两个向量进行精确点积；如此大的长累加器旨在涵盖所有极端情况，避免其最高位发生溢出。

算法 1: 针对该算法的无误差变换
两个浮点数的求和。

输入： a，b 是两个浮点数。
输出： r，s 分别是结果和误差。
函数 $[r, s] =$ `twosum`$(a,\ b)$
  $r := a + b$
  $z := r - a$
  $s := (a - (r - z)) + (b - z)$

算法 2: 两个浮点数乘积的无误差变换。
两个浮点数的乘积。

输入： a，b 是两个浮点数。
输出： r，s 分别为结果和误差。
函数 $[r, s] =$ `twoprod`(a, b)
  $r := a * b$
  $s :=$ 融合乘加(a，b −r)

ExBLAS 项目 Iakymchuk 等（2015）尝试通过为这些运算构建一种多级方法，开发快速、精确且可复现的 BLAS 库，并针对具有复杂多级内存结构的各种现代架构进行定制。一方面，这种方法旨在实现快速性能，以确保性能与非确定性并行版本相近。另一方面，该方法旨在保留最终舍入到目标格式之前的所有信息，从而保证正确舍入，进而实现可复现性。因此，ExBLAS 将长累加器和 FPE 结合起来，形成算法解决方案，并在各种架构上对其进行高效调优和实现，包括常规 CPU、Nvidia 和 AMD GPU，以及 Intel Xeon Phi 协处理器（详情请参见 Collange 等（2015））。因此，ExBLAS 通过确保正确舍入来保证可复现性。

ExBLAS 的基石是可复现并行归约，它是许多 BLAS 例程的核心。ExBLAS 并行归约依赖采用 t wosum EFT 的浮点展开和长累加器，因此能够实现正确舍入且结果可复现。在实际应用中，后者在每次完整求和过程中仅调用一次，因此累加大型向量时带来的开销很小（不到 8%）。本文关注的是两个向量的点积，这是一项至关重要的基础 BLAS 操作。EXDOT 算法基于可复现并行归约和 twoprod 无误差变换：该算法将 twoprod 无误差变换的结果和误差累加到相同的浮点展开中，然后按照归约方案进行处理。我们推导出其分布式版本，该版本使用两个底层浮点展开（一个用于结果，另一个用于误差），并在计算结束时将二者合并。这些以及矩阵-向量乘法、三角求解和矩阵-矩阵乘法等其他例程，均已在 ExBLAS 库† 中实现分布式版本。

3 可复现 Krylov 求解器的通用框架

本节概述了一个通用框架，用于推导任意传统 Krylov 子空间方法的可复现版本。该框架基于两个主要概念：1）识别并行化所引发的问题，以及由此导致的浮点计算非结合性；2）主要借助计算机算术技术和编程指南，谨慎地缓解这些问题。Iakymchuk 等人（2020a、b）在推导预条件共轭梯度法（PCG）的可复现变体时，隐式地采用了该框架。

该框架假定并行平台由 K 个进程（或 MPI 进程）组成，记为 P_1、P_2、...、P_K。其中，系数矩阵 A 被划分为 K 个行块（A_1、A_2、...、A_k），每个 P_k 存储一个行块，其中包含第 k 个分布块 $A_k \in \mathbb{R}^{p_k \times n}$，以及 $n = \sum_{k=1}^{K} p_k$。此外，向量也按照与 A 相同的方式进行划分和分布。例如，残差向量 r 被划分为 r_1、r_2、...、r_K，而 r_k 存储在 P_k 中。此外，标量会在所有 K 个进程上进行复制。

3.1 识别不可复现性的来源

第一步是识别并行环境中导致 Krylov 子空间方法不具结合性、进而不可复现的来源。如图 1 所示，有四种常见操作及其相关的消息传递通信模式：稀疏矩阵-向量乘法（SPMV）以及用于收集向量‡的 Allgatherv；使用全归约集体操作的点积；对向量进行缩放，随后将两个向量相加（AXPY 和类似 AXPY 的操作）；以及应用预条件器。因此，我们将逐一考察这些操作。

一般而言，并行执行中的浮点运算受到扰动时，无法保证结合律和可复现性。例如，调用 MPI Allreduce() 集合操作并不能确保得到相同的结果（或采用相同的执行路径），因为这取决于数据、网络拓扑以及底层的算法实现。在这些假设下，AXPY（及其类似操作）和稀疏矩阵-向量乘法具有结合律安全性，因为它们是在局部数据片段上本地执行的。预条件器的应用

†ExBLAS 代码库：https://github.com/riakymch/exblas
‡当然，对于带状矩阵或类似的稀疏矩阵，也有更好的替代方案，但对于非结构化稀疏矩阵，使用 MPI Allgatherv 是一种简化的解决方案

当（$\tau > \tau_{\max}$）

Step	操作	核函数	通信
$S1:$	$d := Ap$	SPMV	Allgatherv
$S2:$	$\rho := \beta / < p,\ d >$	点积	全归约
$S3:$	$x := x + \rho p$	AXPY	–
$S4:$	$r := r - \rho d$	AXPY	–
$S5:$	$y := M^{-1} r$	应用预条件器	取决于
$S6:$	$\alpha := \beta$	标量运算	–
$S7:$	$\beta := y' * r$	点积	全归约
$S8:$	$\tau := \sqrt{< r,\ r >}$	点积 +平方根	全归约
$S9:$	$\alpha := \beta / \alpha$	标量运算	–
$S10:$	$p := y + \alpha p$	类似 AXPY 的操作	–

结束 while

图 1。 标注了基础线性代数子程序内核和消息传递通信的预条件共轭梯度法。

也可以视为安全操作，例如雅可比预处理器，只要所有操作都不涉及归约；更复杂的预条件器肯定会带来问题。因此，非确定性的主要问题源自并行归约（图 1 中的步骤 $S2$、 $S7$ 和 $S8$）。

3.2 确保可复现性

我们构建了确保可复现性的方法，主要针对点积和并行归约。请注意，Krylov 子空间方法的非确定性实现会调用 BLAS 库（如 Intel MKL）中的点积例程，然后执行 MPI Allreduce()。因此，我们建议将这一过程细化为四个步骤：

- 利用 ExBLAS 及其更轻量的基于 FPE 的版本，构建可复现且正确舍入的点积；

- 通过使用 MPI Allreduce()，将 ExBLAS 和基于 FPE 的点积扩展到分布式内存环境。该集合通信操作作用于长累加器或浮点展开。对于 ExBLAS 方法，我们采用常规归约，因为长累加器是由长整数组成的数组。请注意，在归约 $2 * 2^{K-1}$ 长累加器之后，我们可能需要进行额外的中间归一化，其中 $K = 64 - 52 = 12$ 表示长累加器每个数字所含的无进位位数。对于 FPE 方法，我们定义了基于 TwoSum EFT 的 MPI 操作。因此，此时 MPI Allreduce() 底层所采用的归约算法不会影响计算，因为所有位的信息都得以保留；

- 舍入为双精度：对于长累加器，我们使用 ExBLAS 原生的 Round() 例程。为保证基于 FPE 的计算得到正确舍入结果，我们采用 Rump 等人（2008 年）提出的 NearSum 算法。值得一提的是，舍入操作在本地执行，无需任何通信。在代码的早期版本中，如 Iakymchuk 等人（2022 年）的版本，我们将归约分为三个步骤：MPI Reduce()、舍入和 MPI Bcast()。然而，这种开销可以忽略，因为我们重新确保了对归约操作的控制，从而消除了使用两个集合通信并增加一次同步所带来的性能损失。

显然，ExBLAS 点积提供的结果既经过正确舍入，又具有可复现性。对于轻量级点积，我们还希望它具有通用性，因此提供了采用大小为八的浮点展开并使用提前退出技术的实现。这样，使用浮点展开进行计算时的工作精度可提高至 $8 * 52$ 位，正如 Hida 等人（2001）在双双精度算术中所述。此外，我们在基于浮点展开的实现中增加了一项检查，以应对条件数和/或动态范围过大、无法保留所有信息位的情况。此时会发出警告，并建议切换到基于 ExBLAS 的实现。不过请注意，这些轻量级实现针对条件适中或动态范围适中的问题，旨在兼顾精确性、可复现性和高性能；而 ExBLAS 版本对资源的需求可能很高，尤其是在核心数较少时。总之，如果事先了解问题的相关信息，采用轻量级方法是值得的。

3.3 可编程性方面的工作

需要注意的是，编译器优化，尤其是使用融合乘加（fma）指令时，可能会导致一些非确定性结果。该指令以扩展精度执行 $a * b + c$，并在最后进行单次舍入。例如，在 SPMV 计算中，每个 MPI 进程通过将一块行 A_k 与向量 p相乘，来计算向量 d 中分配给自己的部分 d_k 。由于这些计算在本地按顺序进行，因此具有确定性，也就可以复现。然而，代码中的某些部分，例如 $a * b + c * d * e$ 和 $a+ = b * c$——它们存在于 PBiCGStab 的原始实现中——可能无法始终得出相同结果（Wiesenberger 等，2019）。这是因为，出于性能考虑，C++语言标准允许编译器改变此类运算的执行顺序。它还允许将乘法和求和合并为融合乘加（fma）指令。因此，编译器可能会将 $a * b + c * d$ 转换为两个

乘法 $t1 = a * b$ 和 $t2 = c * d$，随后再进行求和 $t1 + t2$；也可能生成一次乘法$t = c * d$，随后执行 fma(a, b, t)，从而得到略有不同的结果；甚至还可能先计算 $t = a * b$，然后再使用 fma(c, d, t)。因此，我们建议在底层架构支持 fma 的前提下，在 C++11中通过 std::fma 显式指示编译器使用 fma。

另一个重要的观察是，应谨慎地执行除法和数据初始化。例如，Krylov 求解器中 b 的初始化计算为 $b = Ad$，其中 $d = \frac{1}{\sqrt{N}}$取值范围为 $(1, \ldots, 1)^T$。在这种情况下，我们建议先计算 $b = Ad$，其中 $d =$取值范围为 $(1, \ldots, 1)^T$，然后再将 b缩放为 $1/\sqrt{N}$，因为我们观察到 Krylov 求解器的收敛速度略有提高（最高可达 7%）。

4 可复现的双共轭梯度稳定法

经典的双共轭梯度稳定法（BiCGStab）由 van der Vorst（1992）提出，是双共轭梯度法（BiCG）Fletcher（1976）和 CGS Sonneveld（1989）方法的一种快速且收敛平稳的变体。本文介绍预条件 BiCGStab（PBiCGStab）和流水线预条件 BiCGStab（p-PBiCGStab），以及它们结合消息传递接口（MPI）的设计与实现。

对于这两种方法，我们考虑一个线性系统 $Ax = b$，其中系数矩阵 $A \in \mathbb{R}^{n \times n}$ 为稀疏矩阵，含有 n_z个非零元；$b \in \mathbb{R}^n$ 为右端向量；$x \in \mathbb{R}^n$ 为待求解向量。

此外，为简化起见，我们在实现中采用 Saad（2003）提出的雅可比预处理器，该预处理器由矩阵 $M = diag(A)$ 的对角元素构成。因此，预条件器作用于向量，并需要对两个向量进行逐元素乘法。

4.1 消息传递并行 *PBiCGStab* 实现

经典迭代预条件双共轭梯度稳定法的算法描述如图2所示。循环体包含两次稀疏矩阵-向量乘法（$S2$ 和 $S6$）、两次预条件器应用（$S1$ 和 $S5$）、五次点积（$S3$、$S7$、$S10$和 $S11$）、六次 AXPY（类）运算（$S4$、$S8$、$S9$ 和 $S12$），以及若干标量运算（Barrett 等，1994）。

如第 3 节所述，该框架包含 Krylov 子空间方法并行实现中最常见操作的可复现实现。因此，接下来我们将分析 PBiCGStab 求解器消息传递实现中的通信与计算。随后，我们将遵循第 3 节中的指南，推导出可复现版本。

为清晰起见，以下我们将省略变量名中表示迭代次数的上标。因此，例如，x^j 变为 x，后者表示用于存储迭代过程中计算得到的近似序列 x^0、x^1、x^2、…的存储空间。考虑到上述因素，我们分析图2中构成一次 PBiCGStab 迭代循环体的不同计算内核（$S1$–$S12$）。

稀疏矩阵-向量乘积（$S2$、$S6$）：该核函数的输入操作数包括：按行分块分布的系数矩阵 A，以及相应的向量（$\hat{}p$ 或 $\hat{}q$）；该向量按照与 A 相同的分区方式进行分区和分布。为简便起见，下面仅说明如何计算 $S2$。

在计算此核函数之前，我们需要在所有进程中获取分布式向量 $\hat{}p$ 的副本，记为 $\hat{}p \to e$；向量 e 是唯一在所有进程中均有副本的数组。这里可以看出存在一个通信阶段；之后，各进程便可并发计算输出向量 s 的本地部分：P_k：$s_k = A_k\, e$。因此，此核函数需要将向量 $\hat{}p$ 的分布式部分组装成一个在所有进程中均有副本的单一向量 e（例如，在 MPI 中通过 MPI Allgatherv() 实现）。

随后即可并行进行计算，得到处于预期分布式状态的向量结果 s，无需进一步通信。最后，每个 MPI进程都拥有计算所得向量的相应部分。为确保此计算的可复现性，稀疏矩阵 A_k 与 e 的行之间的本地点积基于融合乘加，具体如第 3.3 节所述。

点积（$S3$，$S7$，$S10$，$S11$）：循环体中的下一个核函数是步骤 $S3$ 中分布式向量 r^0 和 s的点积。在这里，每个进程都可以并发计算一个部分结果 P_k：$\rho_k =< r_k^0, s_k >$当所有进程完成这部分计算后，这些中间值必须归约为一个全局复制的标量 α：$= \sigma/(\rho_1 + \rho_2 + \cdots + \rho_K)$。我们可以将相同思路应用于步骤 $S7$、$S10$和 $S11$中的点积，由此总共需要进行五次进程同步（在 MPI 中通过 MPI Allreduce() 实现），因为所有标量都是全局复制的。不过，考虑到 S10 和 S11 中的通信可以合并为一次 MPI Allreduce()，同步次数可以减少到四次。

计算 ρ_k 最简单的方法是调用 Intel MKL 或类似库中的点积例程，但即使内部使用了融合乘加，也无法保证可复现性。因此，我们通过采用两种基于 ExBLAS 的策略来确保可复现性，并遵循第 3.2 节中的指导原则。

AXPY（类似）向量更新（$S4$，$S8$，$S9$，$S12$）：下一个核函数是步骤 $S4$中的类似 AXPY 的核函数，它涉及分布式向量 q，r，s以及全局复制标量 α。步骤 $S8$、$S9$ 和 $S12$ 中的操作遵循相同的思路，因为所有标量都是全局复制的。在这类核函数中，所有进程都可以执行各自的局部计算部分，并在无需任何通信的情况下得到结果：P_k：$q_k = r_k - \alpha\, s_k$。

虽然 AXPY ($y = \alpha x + y$) 可以直接依赖 MKL 库例程，但要实现类似 AXPY 的操作 ($z = \alpha y + x$)，至少需要两个例程（SCAL/ COPY+ AXPY）。为了寻求稳健且正确的解决方案，使用 MKL 例程并非理想选择，因为每个例程都会引入舍入误差。此外，这种方案的开销更大，因为某些向量必须遍历多次。相反，我们建议使用 fma，以单次舍入计算 AXPY 和类似 AXPY 的操作中解向量的每个元素，并且只需遍历向量一次。因此，在可复现版本中，我们提供

为 $A \to M$计算预条件子
设置初始猜测 x^0
初始化 $r^0 := b - Ax^0, p^0 := r^0, \tau^0 := \| r^0 \|_2, j := 0$

当 $(\tau^j > \tau_{\max})$

Step	操作	核函数	Comm
$S1:$	$\hat{p}^j \quad := M^{-1}p^j$	应用预条件子	–
$S2:$	$s^j \quad := A\hat{p}^j$	SpMV	Allgatherv
$S3:$	$\alpha^j \quad := < r^0, \ r^j > / < r^0, \ s^j >$	点积	全归约
$S4:$	$q^j \quad := r^j - \alpha^j s^j$	类似 AXPY 的操作	–
$S5:$	$\hat{q}^j \quad := M^{-1}q^j$	应用预条件子	–
$S6:$	$y^j \quad := A\hat{q}^j$	SpMV	Allgatherv
$S7:$	$\omega^j \quad := < q^j, y^j > / < y^j, y^j >$	两个点积	全归约
$S8:$	$x^{j+1} := x^j + \alpha^j \hat{p}^j + \omega^j \hat{q}^j$	两个 AXPY	–
$S9:$	$r^{j+1} := q^j - \omega^j y^j$	类似 AXPY 的操作	–
$S10:$	$\beta^j \quad := \frac{<r^0, r^{j+1}>}{<r^0, r^j>} * \frac{\alpha^j}{\omega^j}$	点积	全归约
$S11:$	$\tau^{j+1} := \| r^{j+1} \|_2$	点积 +平方根	全归约
$S12:$	$p^{j+1} := r^{j+1} + \beta^j \ (p^j - \omega^j s^j)$	两个类似 AXPY 的操作	–

结束 while 循环

图2. 标注了计算内核和通信的 PBiCGStab 求解器公式。阈值 $\tau_{\max}$ 是计算近似解的相对残差上界。按照记号，$< \cdot, \ \cdot >$ 计算其向量参数的点积（内积）。

自有的 SPMV、AXPY 和类似 AXPY 的操作实现（不依赖任何外部 BLAS 库），从而能够全面控制计算，确保正确舍入和可复现性。

应用预条件器（$S1$, $S5$）：步骤 S1 中的核函数包括应用雅可比预处理器 M，即用矩阵的对角元素对向量 p 进行缩放。因此，所有进程都可以并行执行该操作，因为每个进程都存储一组不同的对角元素（与其存储的矩阵部分相对应）以及相应的一组向量元素：P_k：$\hat{p}_k = M_k^{-1} p_k$。也可以在步骤 $S5$ 中采用相同的流程对向量 q进行缩放，得到 $\hat{q}$。

MKL 库中没有用于计算两个向量逐元素乘积的例程，因此必须采用专门实现。若此代码基于融合乘加，且运算顺序具有确定性，则可确保可复现性，如第 3.3 节所述。

4.2 消息传递并行 *p-PBiCGStab* 实现

Cools 和 Vanroose（2017）提出了推导 Krylov 子空间方法流水线版本的两个主要步骤：

- 通信规避。通过重新排列原始递推关系，减少全局通信的次数。通常，新的递推关系中会包含更多项，因此也会有更多向量运算。

- 隐藏通信。由于在大规模场景下，全局通信是 Krylov 子空间方法中最耗时的组成部分，因此，降低其对并行实现性能影响的另一种方法，是让通信与稀疏矩阵-向量乘法同时执行（重叠）。这种技术通过非阻塞集合通信实现，例如 MPIIallreduce()；需要使用 MPIWait() 检查通信何时完成。

流水线预条件双共轭梯度稳定法（p-PBiCGStab）的算法描述如图 3 所示。循环体包括两次 SPMV（$S10$和 $S18$）、两次预条件器应用（$S9$ 和 $S17$）、六次 DOT 乘积（$S8 \cup S11$ 和 $S16 \cup S19$）、十八次 AXPY/类似 AXPY 的操作（$S1$-$S7$ 和 $S12$-$S15$），以及少量标量运算（Cools 和 Vanroose，2017）。值得一提的是，与标准 PBiCGStab 相比，流水线 PBiCGStab 可能表现出不同的收敛行为，这是由于浮点运算的执行方式不同，因此舍入误差的传播和累积方式也有所不同。

该算法计算核函数的分析与上文第 4.1 节中描述的 PBiCGStab 并行化非常相似。唯一的区别在于点积的实现方式。

点积（$S8 \cup S11$, $S16 \cup S19$）：尽管图 3 中有六个点积，但只需要进行两次全局同步，因为每个步骤中都完成了不止一次归约。因此，在启动同步之前，必须先在每个进程中本地计算与相应归约相关的部分结果。随后，将所得值存储在局部向量中，并使用集合通信计算全局值。重叠执行需要使用非阻塞集合通信，将每次归约分解为两个步骤：第一步（$S8$ 和 $S16$）正确执行 MPI Iallreduce()，启动全局通信；全局通信会在执行其他步骤（例如 $S9$和 $S10$）的同时继续进行。当需要使用全局值时，必须执行第二步，调用 MPI Wait()，因为只有在全局通信

完成后，执行才能继续。这里我们遵循非阻塞通信的“黄金法则”——数据一旦可用就立即启动，并在数据即将被需要之前等待。

5 实验结果

本节报告了多种数值实验，用于考察 PBiCGStab 和 p-PBiCGStab 的原始版本及两个可复现版本的收敛性、可扩展性、精度和可复现性。在实验中，我们采用 IEEE 754 双精度运算，并在弗劳恩霍夫的 SkyLake 分区上进行实验，使用双路 Intel Xeon Gold 6132 CPU（@2.6 GHz）、28 个核心和 192 GB 内存。节点通过 54 Gbit/s 的 FDR InfiniBand 连接。

5.1 在 *SuiteSparse* 矩阵上的评估

我们在单个 SkyLake 节点上，使用 1、2、4、8、16 和 28 个（全部）核心，对 SuiteSparse 矩阵集合中的一系列不同线性系统进行了测试。表2列出了所测试的矩阵，以及行数/列数 N 和非零元数量 nnz。我们的目标是展示算法实现应用于具有不同负载（即非零元数量）和不同复杂度的矩阵时的可复现性、精度和性能。迭代求解器中的右端向量 b 始终初始化为乘积 Ad, $d = \frac{1}{\sqrt{N}}(1, ..., 1)^T$，其中 N 是 A的行数/列数。然而，在表中标记为 ReproPBiCGStab 的 ExBLAS 和基于 FPE 的版本中，我们计算了 $b = Ad$, $d =(1, ..., 1)^T$，然后将 b缩放为 $\frac{1}{\sqrt{N}}$。在所有实现中，迭代均从初始猜测 $x_0 = 0$开始。控制迭代过程收敛的参数为 $\|r^j\|_2/\|r^0\|_2 \leq 10^{-6}$。

表2报告了 PBiCGStab 和 ReproPBiCGStab 达到停止准则所需的迭代次数以及最终真实残差；后者同时标示了基于 ExBLAS 和 FPE 的变体，因为无论使用多少核心/MPI 进程，它们报告的结果都相同。我们还报告了初始残差（$\|r_0\|_2$），结合最终真实残差，可以将其作为判断收敛过程的指标。对于原始版本，我们显示了使用单精度和 16 个核心时的迭代次数，因为二者有所不同。值得注意的是，这两种可复现变体表现出一定趋势：近似结果的精度（最终真实残差）略高且更可靠，和/或收敛速度更快。例如，对于 orsreg 1、rdb3200l、Transport 和 tmt unsym 矩阵，可复现变体所需的迭代次数显著更少；对于另外六个矩阵，两者都略快；只有两个矩阵的收敛速度略慢。对于像vas stokes 2M 这样非常大的矩阵，标准的非确定性 PBiCGStab 方法在使用 16 或 28 个 MPI 进程时无法收敛。

表3展示了流水线 PBiCGStab 及其可复现变体的类似结果。可复现变体往往收敛更快、能够提供更可靠精度的趋势依然存在。例如，对于 orsreg 1、tmt unsym 和 vas stokes 2M 矩阵，两种变体的速度都明显更快；对于另外五个矩阵，两者都略快，而对于四个矩阵则略慢。使用流水线 PBiCGStab，我们能够收敛到在容差为 10^{-6} 时 vas stokes 2M 的近似解。当所需容差提高到例如 10^{-8} 或更高时，标准方法，尤其是流水线方法，可能无法对 vas stokes 2M、ML Geer 和 tmt unsym 收敛。我们将此留待未来研究，并计划探究所需精度与求解器能力之间的这种关联，可能会采用残差替换等修复技术。

图 4 展示了标准 PBiCGStab 方法和流水线 PBiCGStab 方法在每次迭代时计算的残差所对应的收敛历史。图中所示的两个矩阵，即 orsreg 1 和 tmt unsym，代表了可复现变体表现较好的情形：与非确定性变体相比，它们用显著更少的迭代次数便达到了近似解。事实上，这些结果展示了一种理想情形：尽管可复现变体的每次迭代计算开销更高，却能更快收敛至解。此外，在这些可能并不普遍、且会有所波动的情况下，标准 PBiCGStab 和流水线 PBiCGStab 在 16/28 个 MPI 进程上需要更多迭代次数。

图5 展示了 Transport 和 rdb3200l 矩阵上，原始变体以及基于 ExBLAS 和 FPE 的标准 PBiCGStab 与流水线 PBiCGStab 变体的强可扩展性结果——即问题规模固定，而分配的资源数量有所变化。图中报告了求解器整个循环在五次采样中的平均执行时间。我们选择这些矩阵，是因为它们的非零元数量差异很大：Transport 有数百万个，而 rdb3200l 有数千个。请注意，MPI通信在节点内进行，很可能通过共享内存实现节点内通信。对于 Transport，三个变体均表现出良好的可扩展性：在 16 个 MPI 进程上，原始、FPE 和 ExBLAS 变体的标准 PBiCGStab（流水线 PBiCGStab）加速比分别为 8.1x（9.0x）、12.0x（12.7x）和 12.4x（13.0x）；对于 rdb3200l，相应的加速比分别为原始变体 4.2x（5.3x）、FPE 变体 9.1x（11.0x）和 ExBLAS 变体 8.6x（10.9x）。由于增加了浮点运算，可复现变体表现出更高的加速比。在 28 个 MPI 进程上，ExBLAS 和 FPE 变体相对于原始变体的开销，在这两个矩阵上均降至近 2 倍。表2和表3中的其他矩阵也呈现出类似的可扩展性模式和开销。然而，非零元数量较少会导致可扩展性更差、开销更高，rdb3200l 的结果证实了这一点。例如，对于 orsreg 1 矩阵，在 16 个 MPI 进程上，原始变体和 ExBLAS/FPE 变体的速度分别仅为 4 倍和 8 倍。

请注意，对于表 2 和表 3 中的许多矩阵而言，每次循环的平均执行时间并不适合分布式内存计算。这是因为，增加节点带来的潜在性能提升会被通信开销抵消。

5.2 可扩展性

我们采用由三维泊松方程的有限差分法产生的稀疏对称正定系数矩阵，该方程具有 27个模板点。我们将中心点下方的值扰动为 $1.0 - 0.0001$，从而构造出非对称 27点模板，也称为 e型模型（Cools

为 $A \to M$计算预条件子
设置初始猜测 x^0
初始化 $r^0 := b - Ax^0$, $\hat{r}^0 := M^{-1}r^0$, $w^0 := A\hat{r}^0$, $\hat{w}^0 := M^{-1}w^0$, $t^0 := A\hat{w}^0$, $\alpha^0 := < r^0, r^0 > / < r^0, w^0 >$, $\beta^{-1} := 0$, $j := 0$

当 $(\tau^j > \tau_{\max})$

Step	运算	核函数	Comm
$S1:$	$\hat{p}^j := \hat{r}^j + \beta^{j-1}(\hat{p}^{j-1} - \omega^{j-1}\hat{s}^{j-1})$	两个类似 AXPY 的操作	–
$S2:$	$s^j := w^j + \beta^{j-1}(s^{j-1} - \omega^{j-1}z^{j-1})$	两个类似 AXPY 的操作	–
$S3:$	$\hat{s}^j := \hat{w}^j + \beta^{j-1}(\hat{s}^{j-1} - \omega^{j-1}\hat{z}^{j-1})$	两个类似 AXPY 的操作	–
$S4:$	$z^j := t^j + \beta^{j-1}(z^{j-1} - \omega^{j-1}v^{j-1})$	两个类似 AXPY 的操作	–
$S5:$	$q^j := r^j - \alpha^j s^j$	类似 AXPY 的操作	–
$S6:$	$\hat{q}^j := \hat{r}^j - \alpha^j \hat{s}^j$	类似 AXPY 的操作	–
$S7:$	$y^j := w^j - \alpha^j z^j$	类似 AXPY 的操作	–
$S8:$	$< q^j\ y^j >,\ < y_j\ y^j >$	两个点积	Iallreduce 全局归约
$S9:$	$\hat{z}^j := M^{-1}z^j$	应用预条件子	–
$S10:$	$v^j := A\hat{z}^j$	SPMV	Allgatherv
$S11:$	$\omega^j := < q^j, y^j > / < y^j, y^j >$	两个点积	Wait
$S12:$	$x^{j+1} := x^j + \alpha^j \hat{p}^j + \omega^j \hat{q}^j$	两次 AXPY	–
$S13:$	$r^{j+1} := q^j - \omega^j y^j$	类似 AXPY 的操作	–
$S14:$	$\hat{r}^{j+1} := \hat{q}^j - \omega^j(\hat{w}^j - \alpha\hat{z}^j)$	两个类似 AXPY 的操作	–
$S15:$	$w^{j+1} := y^j - \omega^j\ (t^j - \alpha^j v^j)$	两个类似 AXPY 的操作	–
$S16:$	$< r^0\ r^{j+1} >,\ < r^0 w^{j+1} >$ $< r^0, s^j >, < r^0, z^j >$	四次点积	Iallreduce 全局归约
$S17:$	$\hat{w}^{j+1} := M^{-1}w^{j+1}$	应用预条件子	–
$S18:$	$t^{j+1} := A\hat{w}^{j+1}$	SPMV	Allgatherv
$S19:$	$\beta^j := \frac{< r^0, r^{j+1} >}{< r^0, r^j >} * \frac{\alpha^j}{\omega^j}$ $\alpha^{j+1} := \frac{< r^0, r^{j+1} >}{< r^0, w^{j+1} > + \beta^j < r^0, s^j > - \beta^j \omega^j < r^0, z^j >}$	四次点积	Wait

结束循环

图 3. 流水线 PBiCGStab 求解器的公式，标注了计算内核和通信。阈值 $\tau_{\max}$ 是计算近似解相对于解的相对残差上限。在符号表示中，$< \cdot,\ \cdot >$ 计算其向量参数的点积（内积）。

和 Vanroose（2017））。由于 SPMV内核中涉及的向量必须在所有 MPI 进程上复制，这限制了可求解的最大问题规模。鉴于 PBiCGStab 的理论开销为 $t_c \approx 4nnz + 26n$ 浮点算术运算，其中 nnz 表示原始矩阵的非零元数量，而其大小为 n，该方法的执行时间通常由 SPMV内核的执行时间主导。因此，为了分析该方法的弱可扩展性，我们保持每个节点的非零项数量不变。为此，我们对原始矩阵进行了修改，将其转换为带状矩阵，其中下带宽和上带宽（*bandL* 和 *bandU*）分别取决于实验中使用的节点数，具体如下：

$$bandL = bandU = 100 \times \#nodes \quad \to$$
$$nnz = (bandL + bandU + 1) \times n.$$

在 32 个节点上，带宽范围为 100 到 3200。采用这种方法，我们可以保持矩阵的行数/列数等于 n=4M (4,019,679)，同时增加其带宽，从而按比例增加计算工作负载，以匹配硬件资源；这正是弱扩展实验所要求的。

迭代求解器中的右端向量 b 始终初始化为 A 与一个仅包含 1 的向量的乘积；PBiCGStab 迭代则以初始猜测 $x_0 = 0$启动。用于控制迭代过程收敛的参数设为10^{-6}。

图6报告了可复现变体与原始版本在强扩展和弱扩展方面的结果。对于强扩展，我们将问题规模固定为 1600 万个非零元，并改变所用节点数/核心数；对于弱扩展，则通过根据所涉及的节点数调整带宽，使每个节点的工作负载保持为 400 万个非零元；推测负载足够大，可以掩盖通信的影响。在这两种可扩展性测试中，初始开销相同：使用 ExBLAS 的版本为 67%，使用 FPE 的版本为 38%–40%。在强扩展测试中，随着通信开始占据主导地位，两个版本之间的开销差距缩小，ExBLAS 的开销降至 8.2%，FPE 的开销降至 3.0%。然而，弱扩展测试并未呈现这一趋势：ExBLAS 版本的 PBiCGStab 在 32 个节点上显示出约 38% 的开销。

5.3 精度与可复现性

此外，我们还推导出图2所示的 PBiCGStab 顺序版本。该版本依赖 GNU 多精度浮点可靠运算库（MPFR）Fousse 等人（2007）——一个用于在 CPU 上执行多精度（任意精度）浮点计算的 C 库——作为高精度参考实现。此实现使用 2,048 位精度计算点积，使用 192 位精度计算内部逐元素乘积，并将计算结果正确舍入为双精度。

矩阵	Prec	N	nnz	$\|r^0\|_2$	PBiCGStab			ReproPBiCGStab	
					第1次迭代	第16次迭代	$\|b-Ax^j\|_2$	iter	$\|b-Ax^j\|_2$
32位加法	Jac	4,960	19,848	8.00e-03	36	35	4.97e-09	35	7.12e-09
af_shell10	Jac	1,508,065	52,259,885	1.48e+05	9	9	2.18e-02	9	2.18e-02
atmosmodd	Jac	1,270,432	8,814,880	3.75e+03	221	256	2.68e-03	222	9.55e-04
atmosmodj	Jac	1,270,432	8,814,880	3.75e+03	227	227	3.46e-03	229	3.25e-03
atmosmodl	Jac	1,489,752	10,319,760	1.85e+04	133	133	1.80e-02	132	1.68e-02
atmosmodm	Jac	1,489,752	10,319,760	3.50e+05	77	77	2.43e-01	75	2.41e-01
audikw_1	Jac	943,695	77,651,847	1.58e+07	11	11	8.14e+00	11	8.04e+00
bcsstk18	Jac	11,948	149,090	2.30e+09	7	7	7.51e+02	7	7.51e+02
bcsstk26	Jac	1,922	30,336	6.16e+09	11	11	5.62e+03	11	5.62e+03
bone010	Jac	986,703	47,851,783	8.55e+03	12	12	5.91e-03	12	5.91e-03
boneS10	Jac	914,898	40,878,708	7.17e+03	12	12	3.92e-03	12	3.92e-03
Bump_2911	Jac	2,911,419	127,729,899	1.91e+14	12	12	1.82e+08	12	1.82e+08
cage14	Jac	1,505,785	27,130,349	1.00e+00	5	5	1.55e-07	5	1.55e-07
cage15	Jac	5,154,859	99,199,551	1.00e+00	6	6	1.12e-07	6	1.12e-07
circuit5M 直流	Jac	3,523,317	14,865,409	1.02e+04	5	5	6.52e-03	5	6.52e-03
旋度-旋度 3	Jac	1,219,574	13,544,618	2.42e+10	17	17	2.14e+04	17	2.14e+04
旋度-旋度 4	Jac	2,380,515	26,515,867	2.10e+10	19	19	1.18e+04	19	1.18e+04
生态学1	Jac	1,000,000	4,996,000	1.96e+01	8	8	7.61e-06	8	9.66e-06
生态学2	Jac	999,999	4,995,991	1.96e+01	8	8	7.38e-07	8	9.67e-07
Hardesty1	Jac	938,905	12,143,314	9.99e+00	17	20	9.28e-06	19	4.60e-06
ML_Geer	Jac	1,504,002	110,686,677	4.89e+02	2886	3084	2.83e-04	3060	1.19e-04
orsreg_1	Jac	2,205	14,133	4.83e+00	225	243	4.18e-06	210	4.68e-06
女王 4147	Jac	4,147,110	316,548,962	1.94e+14	52	51	3.61e+07	51	7.81e+07
rdb3200l	Jac	3,200	18,880	9.96e+00	641	635	4.10e-06	583	3.17e-06
s3dkq4m2	Jac	90,449	4,427,725	6.08e+02	23	23	7.27e-05	23	7.27e-05
tmt_unsym	Jac	917,825	4,584,801	6.45e-06	6489	6757	9.34e-12	5388	1.20e-11
Transport	Jac	1,602,111	23,487,281	2.45e-02	561	583	2.12e-08	557	1.74e-08
vas_stokes_2M	Jac	2,146,677	65,129,037	4.19e-01	6411	-	3.34e-07	6664	3.49e-07

表2。 PBiCGStab 及其可复现版本（ReproPBiCGStab，二者报告的结果相同）在一组 SuiteSparse 矩阵上的收敛情况。初始猜测为 $x^0 = 0$。在同时报告缩放残差（即 $\|r^j\|_2/\|r^0\|_2$）及相应的真实残差 $\|b - Ax^j\|_2$。

表4报告了在容差为 10^{-6} 、使用八个 MPI 进程时，PBiCGStab 求解器对 orsreg 1 矩阵进行每次迭代所得的中间缩放残差和最终缩放残差（原始代码除外，因为其运行时间较长），与表2中的结果相同。我们还加入了原始代码在单核/单进程上的结果，以凸显可复现性问题。结果以十六进制表示列出所有数位。我们仅报告了少数几次迭代，但差异存在于所有迭代中。顺序 MPFR 版本的结果表明，并行 ExBLAS 和 FPE 变体具有准确性和可复现性：它们报告的迭代次数、中间残差、最终真实残差以及初始缩放残差均完全一致。不过，PBiCGStab 的 MPFR 变体在 3.39e-01 秒内收敛至近似解，而 ExBLAS 和 FPE 变体分别耗时 3.95e-02 秒和 2.75e-02 秒（分别快 8.57 倍和 12.32 倍），均使用八个 MPI 进程。原始代码从初始迭代时的少数几位，到最终迭代时几乎整个数值，都出现了差异；所需的迭代次数也与可复现版本和 MPFR 变体不同。

6 相关工作

为增强可复现性，Intel 在其数学核心库（MKL）中提出了“条件数值可复现性”（CNR）选项。尽管 CNR 能保证可复现性，但无法确保正确舍入，这意味着其精度存在争议。此外，使用 CNR 获取可复现结果的成本很高。例如，对于大型数组，在索邦大学托管的 Mesu集群上，采用 CNR 的 MKL 加法速度几乎比常规 MKL 加法慢 2 倍（Collange 等（2015））。

Demmel 和 Nguyen 基于 Rump、Ogita 和 Oishi 的研究成果 Rump et al. (2010, 2008)，实现了一系列用于浮点运算中可复现求和的算法 Demmel and Nguyen (2013, 2015)。这些算法总是返回相同的结果。它们首先计算总和的绝对界，然后将所有数舍入到该界的某个分数。由此，舍入后各数的加法是精确的；然而，使用他们的实现和两个或三个分箱计算出的总和并未正确舍入。他们的结果显示，与使用 1024 个处理器

矩阵	Prec	N	nnz	$\|r^0\|_2$	p-PBiCGStab			p-ReproPBiCGStab	
					第1次迭代	第16次迭代	$\|b-Ax^j\|_2$	iter	$\|b-Ax^j\|_2$
32位加法	Jac	4,960	19,848	8.00e-03	36	35	3.88e-07	35	6.21e-07
af_shell10	Jac	1,508,065	52,259,885	1.48e+05	9	9	2.18e-02	9	2.18e-02
atmosmodd	Jac	1,270,432	8,814,880	3.75e+03	222	221	1.66e-03	241	3.55e-03
atmosmodj	Jac	1,270,432	8,814,880	3.75e+03	220	223	2.92e-03	223	3.53e-03
atmosmodl	Jac	1,489,752	10,319,760	1.85e+04	140	134	1.06e-02	132	1.82e-02
atmosmodm	Jac	1,489,752	10,319,760	3.50e+05	77	78	3.37e-01	77	2.55e-01
audikw_1	Jac	943,695	77,651,847	1.58e+07	11	11	7.03e+00	11	8.04e+00
bcsstk18	Jac	11,948	149,090	2.30e+09	7	7	7.51e+02	7	7.51e+02
bcsstk26	Jac	1,922	30,336	6.16e+09	11	11	5.62e+03	11	5.62e+03
bone010	Jac	986,703	47,851,783	8.55e+03	12	12	5.91e-03	12	5.91e-03
boneS10	Jac	914,898	40,878,708	7.17e+03	12	12	3.92e-03	12	3.92e-03
Bump_2911	Jac	2,911,419	127,729,899	1.91e+14	12	12	1.82e+08	12	1.82e+08
cage14	Jac	1,505,785	27,130,349	1.00e+00	5	5	1.55e-07	5	1.5e-07
cage15	Jac	5,154,859	99,199,551	1.00e+00	6	6	1.12e-07	6	1.12e-07
circuit5M 直流	Jac	3,523,317	14,865,409	1.02e+04	5	5	6.52e-03	5	6.52e-03
旋度-旋度 3	Jac	1,219,574	13,544,618	2.42e+10	17	17	2.14e+04	17	2.14e+04
旋度-旋度 4	Jac	2,380,515	26,515,867	2.10e+10	19	19	1.18e+04	19	1.18e+04
生态学1	Jac	1,000,000	4,996,000	1.96e+01	9	8	1.34e-06	8	9.60e-06
生态学2	Jac	999,999	4,995,991	1.96e+01	8	8	7.18e-06	9	1.07e-05
Hardesty1	Jac	938,905	12,143,314	9.99e+00	17	18	5.13e-06	20	2.13e-06
ML_Geer	Jac	1,504,002	110,686,677	4.89e+02	2426	4429	5.64e-02	3420	5.83e-02
orsreg_1	Jac	2,205	14,133	4.83e+00	239	279	1.09e-06	175	3.03e-06
女王 4147	Jac	4,147,110	316,548,962	1.94e+14	52	53	6.02e+07	51	5.40e+07
rdb32001	Jac	3,200	18,880	9.96e+00	671	615	9.79e-06	685	8.41e-06
s3dkq4m2	Jac	90,449	4,427,725	6.08e+02	23	23	7.27e-05	23	7.27e-05
tmt_unsym	Jac	917,825	4,584,801	6.45e-06	6641	7996	9.76e-06	4999	3.01e-10
Transport	Jac	1,602,111	23,487,281	2.45e-02	580	600	5.34e-09	582	2.32e-08
vas_stokes_2M	Jac	2,146,677	65,129,037	4.19e-01	6880	6358	2.77e-07	6232	3.81e-07

表3。 在一组 SuiteSparse 矩阵上，**流水线 PBiCGStab 及其可复现版本**（p-ReproPBiCGStab，两者报告的结果相同）的收敛情况。初始猜测为 $x^0 = 0$。报告了达到缩放残差容差 10^{-6} 所需的迭代次数，即 $\|r^j\|_2/\|r^0\|_2$，以及相应的真实残差 $\|b - Ax^j\|_2$。

迭代次数	残差			
	MPFR	原始1 个进程	原始 8 个进程	*ExBLAS* 和 *FPE*
0	0x1.3566ea57eaf3fp+2	0x1.3566ea57ea**b49**p+2	0x1.3566ea57ea**b49**p+2	0x1.3566ea57eaf3fp+2
1	0x1.146d37f18fbd9p+0	0x1.146d37f18f**aaf**p+0	0x1.146d37f18f**ab**p+0	0x1.146d37f18fbd9p+0
...	...	...	...	...
99	0x1.cedf0ff322158p-13	**0x1.88008701ba87p-12**	**0x1.04e23203fa6fcp-12**	0x1.cedf0ff322158p-13
100	0x1.be3698f1968cdp-13	**0x1.55418acf1af27p-12**	0x1.**fbf5d3a5d1e49**p-13	0x1.be3698f1968cdp-13
...	...	...	...	...
208	0x1.355b0f18f5ac1p-20	**0x1.19edf2c932ab8p-18**	0x1.**b051edae310c7**p-20	0x1.355b0f18f5ac1p-20
209	0x1.114dc7c9b6d38p-20	**0x1.19b74e383f74ep-18**	0x1.**a18fc929018d4**p-20	0x1.114dc7c9b6d38p-20
210	0x1.03b1920a49a7ap-20	**0x1.19c846848f361p-18**	0x1.**c7eb5bbc198b1**p-20	0x1.03b1920a49a7ap-20

表4。 orsreg_1 矩阵的中间残差和最终残差的精度与可复现性（相对于 MPFR），见表2。

相比，开销大约增加了 20%（仅限 CPU），而在 32 个处理器（单个节点）上则会慢 3.4 倍。Ahrens、Nguyen 和 Demmel 将这一概念扩展到其他少数几种可复现 BLAS 例程，并以 ReproBLAS 库的形式发布（http://bebop.cs.berkeley.edu/reproblas/），但仅支持并行可复现归约。此外，ReproBLAS 的工作还扩展到可复现高瘦 QR，参见 Nguyen 和 Demmel（2015）。

另一种确保可复现性的方法称为 ExBLAS，最初由 Collange、Defour、Graillat 和 Iakymchuk 在 Collange 等（2015）中提出。ExBLAS 基于长累加器与

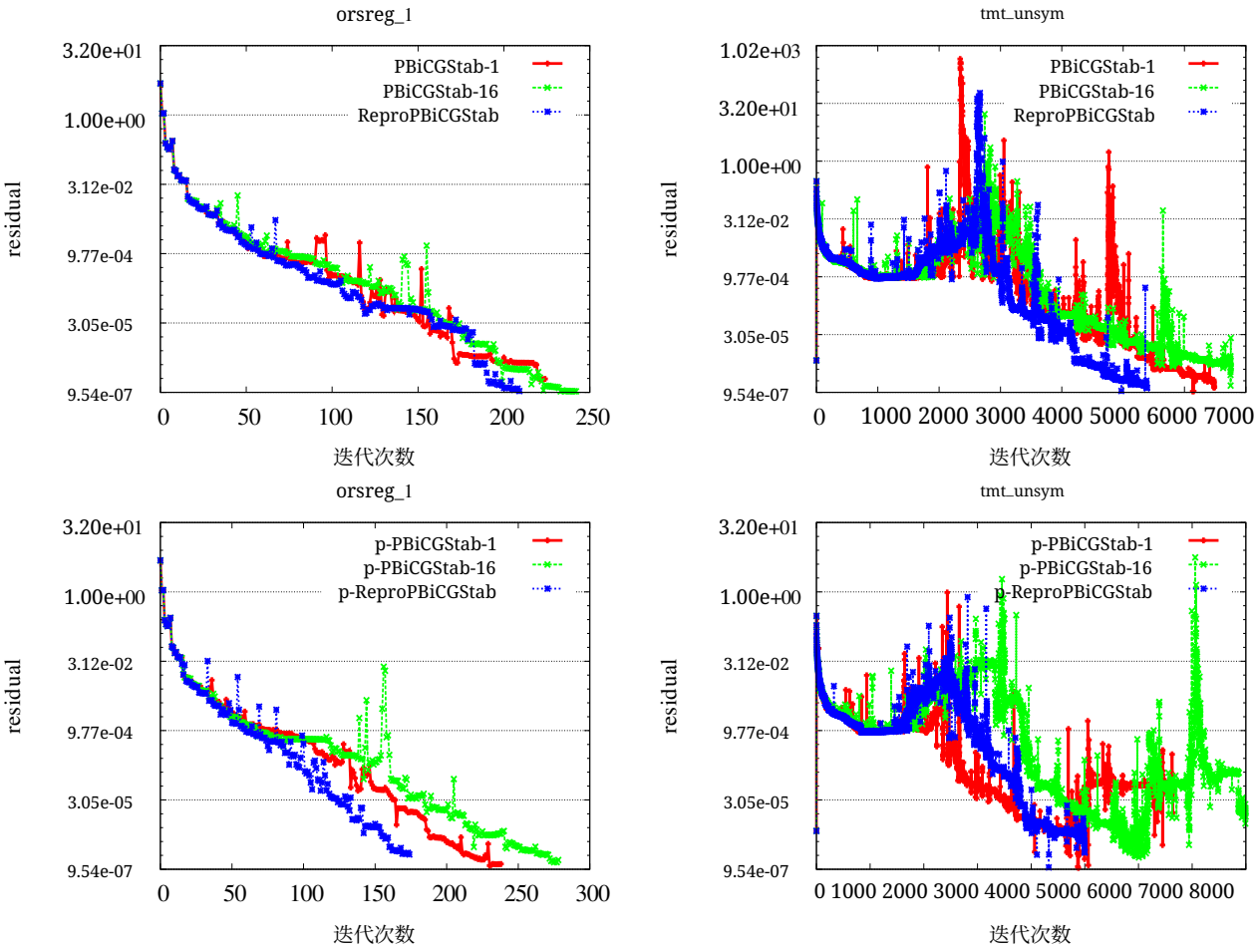

图 4. 标准 PBiCGStab 及其可复现变体（第一行），以及流水线 PBiCGStab 及其可复现变体（第二行）的残差历史；第一列显示 orsreg 1 的结果，第二列显示 tmt unsym 的结果，矩阵的详细信息请参见表2。请注意，最后一次迭代未显示。

浮点展开相结合，并配合无误差变换使用。该方法将在第 2 节介绍。Collange 等人指出，Collange 等（2015）中提出的可复现且准确的求和算法，在 512 个核心（32 个节点）上开销为 8%，在 16 个核心（一个节点）上开销低于 2%。ExSUM 适用于广泛的体系结构以及分布式内存集群，而其他例程主要面向 GPU。利用线性代数算法的模块化和层次化结构，Iakymchuk 等人（2019）将 ExBLAS 方法应用于构造带部分选主元的可复现 LU 分解。

Mukunoki 和 Ogita 提出了实现可复现 BLAS 的方法，称为 OzBLAS Mukunoki 等（2019），其精度可调。该方法与 ReproBLAS 和 ExBLAS 均不同，因为它不需要从头实现每个 BLAS 例程，而是依赖高性能（供应商提供的）实现。因此，OzBLAS 实现了 Ozaki 方案 Ozaki 等（2012），该方案采用分叉-汇合方法：将矩阵和向量拆分（对每个元素进行切片）为子矩阵和子向量，以便在避免溢出的情况下进行安全的乘法；随后，对每个拆分部分调用高性能 BLAS；最后，使用例如 NearSum 算法将结果合并。目前，OzBLAS 库包含点积、矩阵-向量乘法（gemv）和矩阵-矩阵乘法（gemm）。这些算法变体及其在 GPU 和 CPU 上的实现（CPU 上仅支持点积）可确保 BLAS 核函数的可复现性，并且精度可调，最高可达正确舍入结果。

所提出的框架被隐式用于推导可复现的预条件共轭梯度法（PCG）变体，包括扁平 MPI 版本（Iakymchuk 等，2020b）和混合 MPI 与 OpenMP 任务版本（Iakymchuk 等，2020a）。这些可复现的 PCG 变体主要通过三维泊松方程进行验证，该方程采用 27个模板点；结果表明，在 MareNostrum4 集群上最多使用 768 个核心时，其可扩展性良好，性能开销较低（ExBLAS 和轻量级变体均低于 30%）。

7 结论

并行 Krylov 子空间方法在并行环境中实现时可能缺乏可复现性，表 2 至表 4 的结果证实了这一点。调试以及验证与确认都需要数值可靠性。本文提出了一种通用框架，用于在任意 Krylov 子空间方法中重建可复现性并确保精度。我们的框架包含两个步骤：分析底层算法中的数值

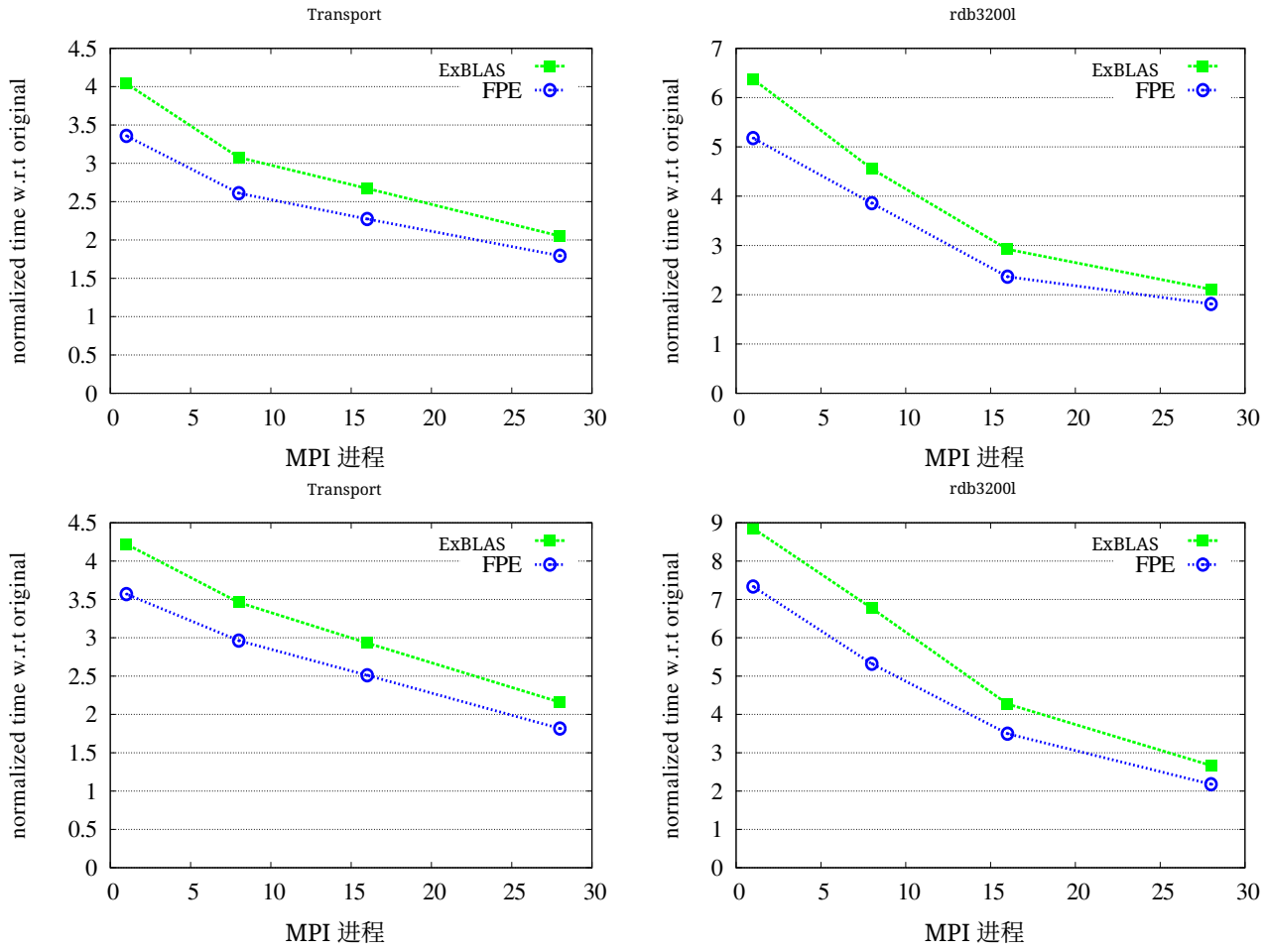

图5. 标准 PBiCGStab 及其可复现变体（第一行），以及流水线 PBiCGStab 及其可复现变体（第二行）在单个 SkyLake 节点上的强扩展结果；第一列显示 Transport 的结果，第二列显示 rdb32001 的结果，矩阵详情见表2。

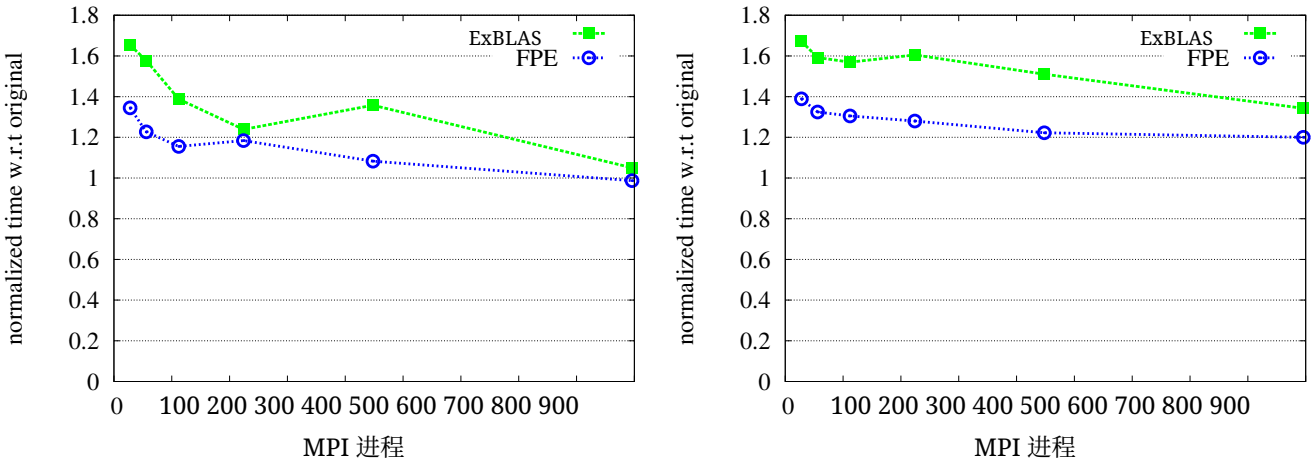

图6. 可复现 PBiCGStab 变体的强扩展（左）和弱扩展（右），归一化时间以非确定性 MPI 变体为基准。

异常；通过算法解决方案和可编程性提示加以处理。算法解决方案基于 ExBLAS 项目构建，具体而言：ExBLAS 将长累加器与浮点展开有效结合；轻量级版本则采用浮点展开。可编程性方面的工作重点在于具体做法包括：显式调用融合乘加指令，避免编译器将其替换；使用定制的、基于融合乘加的 AXPY 和类似 AXPY 的操作，而非 MKL 或类似的 BLAS 库；并将除法推迟到实际需要时再执行。

我们选用预条件标准 BiCGStab 方法和流水线 BiCGStab 方法作为测试用例，并分别为它们各自构造了两种可复现算法变体。值得一提的是，这两种 BiCGStab 方法实际上是不同的算法，它们的操作集合不同，因此计算路径并不相同，舍入误差的传播方式也各异；即使采用可复现变体，这种差异仍可从图 4 的收敛历史中看出。可复现变体使标准方法和流水线 PBiCGStab 均能得到相同的结果，并经其 MPFR 版本验证，从而确保迭代次数、中间残差和最终残差，以及所求解向量均具有可复现性。我们在一组 SuiteSparse 矩阵上验证了实现，结果显示，Ex BLAS 和基于 FPE 的版本的性能开销接近 2.0 倍，后者的开销明显较低。对 32 个节点上的 27 点模板进行的测试表明，两者的开销几乎可以忽略，分别为 8% 和 3%。代码见 https: // github.com/riakymch/ReproPBiCGStab。

通过本研究，我们希望通过适当选择底层库以及细致的编程工作，促进设计层面的可复现性。例如，可以提供如下简要指导：1）对于基础数值计算，使用 ExBLAS、ReproBLAS 或 OzBLAS 等可复现底层库，Mukunoki 等（2019）；2）分析算法，通过消除可能违反结合律的不确定性以及非确定性的计算顺序，使算法具备可复现性，例如归约、融合乘加的使用或不使用，以及将除法推迟到实际需要时执行。此外，我们试图论证迭代求解器需要按位可复现且正确舍入的结果，因为当我们未达到所需容差时，这些结果无论如何都会在后续迭代中得到进一步改进；因此，我们无法充分利用所获得的按位结果。随着混合精度方法的发展，这一点会变得更加明显，而这正是我们计划继续研究的方向。

我们未来的工作是研究流水线 Krylov 子空间求解器中的残差替换策略，例如流水线 PBiCGStab（p-PBiCGStab）Cools 和 Vanroose（2017），并研究长累加器和浮点展开所提供的更高精度能否缓解此类策略带来的问题。我们认为，可以利用长累加器和浮点展开所提供的更高精度，来缓解舍入误差传播方式的差异，并应对 p-PBiCGStab 中可达到精度损失。

致谢

本研究部分由欧盟 H2020 MSCA-IF Robust 项目（编号 842528）、欧盟 H2020 CoE CEEC（编号 101093393）以及法国 ANR InterFLOP 项目（编号 ANR-20-CE46-0009）资助。海梅一世大学的研究由项目 PID2020-113656RB-C21 资助，资金通过 MCIN/AEI/10.13039/501100011033 提供；此外还获得项目 UJI-B2021-58 的资助。

参考文献

Iakymchuk R 等（2020a）《混合编程环境中并行预条件共轭梯度法的可复现性》。*IJHPCA*34(5)：502–518。DOI: 10.1177/ 1094342020932650。

Iakymchuk R 等（2020b）并行预条件共轭梯度法的可复现策略。*JCAM* 371：112697。DOI:10.1016/j.cam.2019.112697。Barrett R 等（1994）线性方程组求解模板：迭代方法的构件，第 *2* 版。SIAM。Collange S 等（2015）多核与众核架构上的并行归约数值可复现性。*ParCo*49：83–97。Cools S 和 Vanroose W（2017）用于并行求解大型非对称线性方程组的通信隐藏流水线 BiCGstab 方法。*ParCo*65：1–20。DOI:10.1016/j.parco.2017.04.005。Demmel J 和 Nguyen HD（2013）快速可复现浮点求和。载于：*ARITH-21* 论文集。第 163–172 页。Demmel J 和 Nguyen HD（2015）并行可复现求和。*IEEE*《计算机汇刊》64(7)：2060–2070。Fletcher R（1976）不定系统的共轭梯度法。载于：Watson GA（编）数值分析。Springer Berlin Heidelberg。ISBN 978-3-540-38129-7，第 73–89 页。Fousse L 等（2007）MPFR：支持正确舍入的多精度二进制浮点库。*ACM TOMS*33(2)：13。DOI:10.1145/1236463.1236468。Goldberg D（1991）每位计算机科学家都应了解的浮点运算知识。*ACMComput.Surv.* 23(1)：5–48。DOI:10.1145/103162.103163。Hida Y、Li XS 和 Bailey DH（2001）四双精度浮点运算算法。载于：*ARITH- 15* 论文集。第 155–162 页。DOI:10.1109/ARITH.2001.930115。Iakymchuk R、Collange S、Defour D 和 Graillat S（2015）ExBLAS：可复现且准确的 BLAS 库。载于：*NRE2015* 研讨会论文集（作为 *SC15* 的一部分举行）。美国德克萨斯州奥斯汀，*2015*年*11*月*15*日至*20*日。第 1–4 页。Iakymchuk R、Graillat S、Defour D 和 Quintana-Orti ES（2019）构建可复现 LU 分解的分层方法。*IJHPCA* ： 1–13（即将发表）。HAL 预印本：hal- 01419813。Iakymchuk R、Graillat S 和 Jos´e A（2022）构建可复现 Krylov 子空间算法的通用框架：一个 bicgstab 案例研究。载于：*PPAM 2022* 论文集。*Springer LNCS*。第 1–12 页。即将发表。预印本见 https://hal.sorbonne-universite.fr/hal-03382119/。IEEE 计算机学会（2008）*IEEE* 浮点算术标准。IEEE 754-2008 标准。Knuth DE（1969）计算机程序设计艺术：半数值算法，第 2 卷。Addison-Wesley。Kulisch U 和 Snyder V（2011）精确点积：长区间算术的基本工具。*Computing* 91(3)：307–313。Mukunoki D、Ogita T 和 Ozaki K（2019）采用 Ozaki 方案、适用于众核架构的准确且可复现 BLAS 例程。载于：*PPAM 2019* 论文集。*Springer LNCS*，第 12043 卷。第 516–527 页。DOI:10.1007/978-3-030-43229-4 44。Nguyen HD 和 Demmel J（2015）可复现高瘦 QR。载于：*ARITH-22* 论文集。第 152–159 页。DOI:10.1109/ARITH.2015.28。Ogita T、Rump SM 和 Oishi S（2005）准确求和与点积。*SIAMJ.Sci.Comput* 26：1955–1988。

Ozaki K、Ogita T、Oishi S 和 Rump SM（2012）利用快速矩阵乘法例程实现矩阵乘法的无误差变换及其应用。*NumericalAlgorithms*59(1)：95–118。Rump SM、Ogita T 和 Oishi S（2008）准确浮点求和第二部分：符号、K 折忠实性与就近舍入。*SIAM J. Sci. Comput.*31(2)：1269–1302。Rump SM、Ogita T 和 Oishi S（2010）快速高精度求和。*Nonlinear Theory and ItsApplications, IEICE*1(1)：2–24。Saad Y（2003）稀疏线性系统的迭代方法。第 2 版。美国宾夕法尼亚州费城：SIAM。Saad Y 和 Schultz MH（1986）GMRES：用于求解非对称线性系统的广义最小残差算法。*SIAM Journal on Scientific and Statistical Computing* 7：856–869。Sonneveld P（1989）CGS：用于非对称线性系统的快速 Lanczos 型求解器。*SIAMJ. Sci. Stat.Comp.* 10(1)：36–52。van der Vorst HA（1992）Bi-CGSTAB：一种快速且平稳收敛的 Bi-CG 变体，用于求解非对称线性系统。*SIAMJ. Sci.Stat. Comp.* 13(2)：631–644。DOI:10.1137/0913035。Wiesenberger M 等（2019）Feltor 代码及其库在并行计算机体系结构上的可复现性、精度和性能。*CPC* 238：145–156。