**引用格式：**卢玥辰, 袁雨萧, 杨德闯, 等. GPU 上 Tensor Core 加速的共轭梯度解法器 [J]. 电子科技大学学报, 2026, 55(2): 244-251.
LU Y C, YUAN Y X, YANG D C, et al. Tensor Core accelerated conjugate gradient solver on GPUs[J]. Journal of University of Electronic Science and Technology of China, 2026, 55(2): 244-251.

# GPU 上 Tensor Core 加速的共轭梯度解法器

卢玥辰，袁雨萧，杨德闯，刘伟峰*

（中国石油大学（北京） 人工智能学院，北京 102249）

**摘要：**共轭梯度方法（CG）和稳定双共轭梯度方法（BiCGSTAB）是求解稀疏线性系统的两种经典且高效的迭代方法，被广泛应用于科学计算和工程问题中。尽管 GPU 等并行处理器提升了这两种方法的并行性，但最新的硬件单元 Tensor Core 及其计算能力尚未被用于这两种方法中。该文设计了一个 Tensor Core 加速的 CG 解法器，利用 Tensor Core 计算 CG 和 BiCGSTAB 方法中的关键组件稀疏矩阵–向量乘法（SpMV）和点积操作，以发挥 Tensor Core 的计算能力，从而提升两种方法的整体性能。在 NVIDIA A100 和 H100 GPU 上的实验结果表明，Tensor Core 加速的这两种方法相比调用 CUDA 官方库的基准版本在多个稀疏矩阵上均取得了显著的加速效果。

**关键词：**稀疏矩阵–向量乘法；点积；共轭梯度法；稳定双共轭梯度法；张量核心；图形处理单元
**中图分类号：**TP317 **文献标志码：**A DOI: 10.12178/1001-0548.2024358

# Tensor Core accelerated conjugate gradient solver on GPUs

LU Yuechen, YUAN Yuxiao, YANG Dechuang, and LIU Weifeng*

(College of Artificial Intelligence, China University of Petroleum-Beijing, Beijing 102249, China)

**Abstract:** Conjugate gradient (CG) and biconjugate gradient stabilized (BiCGSTAB) are two classical and efficient iterative methods for solving sparse linear systems, widely used in scientific computing and engineering applications. Although GPUs and other parallel processors have enhanced the parallelism of these methods, the latest hardware unit, Tensor Core, and its computing power have not yet been fully exploited for these two methods. This work proposes a Tensor Core-accelerated CG solver that leverages Tensor Cores for the key components in the CG and BiCGSTAB methods, such as sparse matrix-vector multiplication (SpMV) and dot product computation, thereby exploiting the computational capability of Tensor Cores to improve the overall performance of both methods. Experimental results on NVIDIA A100 and H100 GPUs demonstrate that both of these methods accelerated by Tensor Core achieve significant speedups over the baseline version that uses the CUDA official library on various sparse matrices.

**Key words:** sparse matrix-vector multiplication; dot product; conjugate gradient; biconjugate gradient stabilized; Tensor Core; GPU

共轭梯度法（conjugate gradient, CG）[1] 和稳定双共轭梯度法（biconjugate gradient stabilized, BiCGSTAB）[2] 是求解大型稀疏线性系统的两种经典且有效的迭代法。CG 方法适用于对称正定矩阵，而 BiCGSTAB 方法则能够处理一般非对称矩阵。由于它们的适用性和有效性，这两种方法在科学计算和工程应用中得到了广泛应用。然而，随着问题规模的日益增大，计算的复杂性和时间消耗也显著增加。因此，如何加速这些算法成为一个关键问题。

随着新型处理器计算能力的不断提升，许多研究致力于在各种现代并行计算平台上加速稀疏矩阵计算和并行求解算法。文献 [3] 针对申威众核架构提出了一系列稀疏矩阵–向量乘法（SpMV）的性能优化策略，文献 [4] 提出了基于稀疏矩阵结构特性的分块并行策略，成功应用于多核平台中。文献 [5−6] 对稀疏矩阵计算自动调优方法进行了深入

收稿日期：2024−12−30
基金项目：国家自然科学基金（U23A20301, 62372467）
作者简介：卢玥辰，博士生，主要从事稀疏矩阵计算方面的研究。
*通信作者 E-mail: weifeng.liu@cup.edu.cn

研究。文献 [7] 和文献 [8] 分别对代数多重网格算法的应用现状、可扩展瓶颈进行了分析。文献 [9] 对稀疏近似逆预条件子的并行计算进行了深入研究，优化了预条件子的构造过程并提高了其在稀疏线性系统求解中的效率。文献 [10] 提出了基于扩展 Krylov 子空间的可扩展线性求解器，文献 [11] 进一步研究了 GPU 上的混合精度 s-step 共轭梯度法。

与上述研究不同，Tensor Core 的引入为稀疏矩阵计算提供了新的可能。Tensor Core 是由 NVIDIA 首次提出并集成到其近几代 GPU[12-14] 中的新型专用硬件单元，旨在加速深度学习和高性能计算中的矩阵运算操作。与传统的计算单元相比，Tensor Core 能够在相同的时间内执行更多的矩阵乘法运算，因而在处理复杂的计算任务时表现出显著的性能优势。Tensor Core 的引入为 GPU 计算带来了革命性的变化，使得通用 GPU 处理复杂计算问题的能力大幅提升，也为优化 CG 和 BiCGSTAB 方法提供了新的可能性。

在 CG 和 BiCGSTAB 方法中，浮点运算操作包括 SpMV、点积（Dot product）计算和向量的数乘和加法（AXPY）运算。为了进一步理解 CG 和 BiCGSTAB 方法的性能表现，本文在 NVIDIA H100 GPU 上分别统计了这两个算法的各个计算过程开销，收集了来自 SuiteSparse 稀疏矩阵集[15] 的 149（CG）和 566（BiCGSTAB）个矩阵的统计结果。可以发现，SpMV 操作和点积计算开销占据了 CG 和 BiCGSTAB 两个算法整体的绝大多数时间：SpMV 操作和点积计算的总开销在两个方法中的平均占比分别达到了 73.06% 和 85.46% 左右。因此，提高 SpMV 操作和点积计算的效率是加速 CG 和 BiCGSTAB 算法的关键。

本文旨在提出一个利用 Tensor Core 加速的 CG 和 BiCGSTAB 方法。使用了 Tensor Core 来加速 CG 和 BiCGSTAB 方法中的关键组件 SpMV 和点积计算：在 SpMV 部分，根据 CG 和 BiCGSTAB 的算法特性，对已有的使用 Tensor Core 加速的 SpMV 算法[16] 进行了调整，使其能够更高效地处理这两种算法中常见的稀疏矩阵模式；对于点积计算部分，根据文献 [17] 提出的利用 Tensor Core 加速 Reduction 操作的思想，本文设计了使用 Tensor Core 加速的点积计算算法。经过调整后，Tensor Core 计算的 SpMV 和点积操作被应用在 CG 和 BiCGSTAB 的计算流程中，从而充分发挥 Tensor Core 的计算优势。

# 1　背景

## 1.1　共轭梯度方法和稳定双共轭梯度方法

算法 1　共轭梯度方法伪代码

Initialize $\boldsymbol{x}_0$

$\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$

$\boldsymbol{p}_0 := \boldsymbol{r}_0$, $\varepsilon := 10^{-10}$, $j := 0$

while $j <$ maxiter and $\|\boldsymbol{r}_j\|_2 > \varepsilon$ do

　$\boldsymbol{\mu} := A\boldsymbol{p}_j$ //稀疏矩阵−向量乘法

　$a_j := (\boldsymbol{r}_j, \boldsymbol{r}_j) / (\boldsymbol{\mu}, \boldsymbol{p}_j)$ //点积

　$\boldsymbol{x}_{j+1} := \boldsymbol{x}_j + a_j\boldsymbol{p}_j$ //向量的数乘和加法

　$\boldsymbol{r}_{j+1} := \boldsymbol{r}_j - a_j\boldsymbol{\mu}$ //向量的数乘和加法

　$\boldsymbol{\beta}_j := (\boldsymbol{r}_{j+1}, \boldsymbol{r}_{j+1}) / (\boldsymbol{r}_j, \boldsymbol{r}_j)$ //点积

　$\boldsymbol{p}_{j+1} := \boldsymbol{r}_{j+1} + \boldsymbol{\beta}_j\boldsymbol{p}_j$ //向量的数乘和加法

　$j := j+1$

end while

共轭梯度方法是一种求解对称正定稀疏矩阵线性方程组的迭代方法[1]。它适用于因规模过大而无法通过直接法处理的稀疏线性系统。共轭梯度法从初始估计 $\boldsymbol{x}_0$ 开始，通过迭代的方式逐渐逼近解。每一步的迭代都朝着一个与之前所有方向共轭的方向进行。其优点是所需存储量小，具有逐步收敛性，稳定性高。

算法 2　稳定双共轭梯度方法伪代码

Initialize $\boldsymbol{x}_0$

$\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{r}_0^* := \boldsymbol{r}_0$

$\boldsymbol{p}_0 := \boldsymbol{r}_0$, $\varepsilon := 10^{-10}$, $j := 0$

while $j <$ maxiter and $\|\boldsymbol{r}_j\|_2 > \varepsilon$ do

　$\boldsymbol{\mu} := A\boldsymbol{p}_j$ //稀疏矩阵−向量乘法

　$a_j := (\boldsymbol{r}_j, \boldsymbol{r}_0^*) / (\boldsymbol{\mu}, \boldsymbol{r}_0^*)$ //点积

　$\boldsymbol{s}_j := \boldsymbol{r}_j + a_j\boldsymbol{\mu}$ //向量的数乘和加法

　$\boldsymbol{\theta} := A\boldsymbol{s}_j$ //稀疏矩阵−向量乘法

　$\omega_j := (\boldsymbol{\theta}, \boldsymbol{s}_j) / (\boldsymbol{\theta}, \boldsymbol{\theta})$ //点积

　$\boldsymbol{x}_{j+1} := \boldsymbol{x}_j + a_j\boldsymbol{p}_j + \omega_j\boldsymbol{s}_j$ //向量的数乘和加法

　$\boldsymbol{r}_{j+1} := \boldsymbol{s}_j - \omega_j\boldsymbol{\theta}$ //向量的数乘和加法

　$\boldsymbol{\beta}_j := (\boldsymbol{r}_{j+1}, \boldsymbol{r}_0^*) / (\boldsymbol{r}_j, \boldsymbol{r}_0^*) \times a_j / \omega_j$ //点积

　$\boldsymbol{p}_{j+1} := \boldsymbol{r}_{j+1} + \boldsymbol{\beta}_j(\boldsymbol{p}_j - \omega_j\boldsymbol{\mu})$ //向量的数乘和加

end while

稳定双共轭梯度方法[2] 克服了共轭梯度算法不适用于非正定或非对称矩阵的局限性，但是它在每次迭代过程中需要更多的计算。它通过引入稳定化技术在每一步迭代中使用两个共轭方向来提高算法的稳定性和加快收敛速度，从而更有效地解决一般性的线性方程组。该方法在解决大规模

线性方程组时具有较高的效率和稳定性，并且可以方便地与各种预处理技术结合使用，以进一步提高求解效率。

算法 1 和算法 2 分别展示了共轭梯度方法和稳定双共轭梯度方法的伪代码，从中可以发现这两种方法主要包含 SpMV、点积计算以及 AXPY 这 3 种操作。

### 1.2 矩阵乘法累加单元

近些年来，现代并行处理器对于矩阵乘法累加（matrix multiply-accumulate, MMA）单元的引入为突破现有计算瓶颈带来了新的希望。NVIDIA 的张量核心（Tensor Core）、AMD 的矩阵核心（Matrix Core）、Apple 的矩阵协处理器（AMX）以及 Intel GPU 上的 Xe 矩阵扩展（XMX）和 CPU 上的高级矩阵扩展（AMX）都是此类 MMA 单元的代表。以 Tensor Core 为例，它是 NVIDIA 专为小型通用矩阵乘法（general matrix multiplication, GEMM）设计的计算单元，Ampere 架构中的每个 Tensor Core 可以在一个时钟周期内同时处理 64 个 FP16 运算或 32 个 TF32 运算。到目前，第 4 代 Tensor Core 已经能够支持低精度整型（INT8）、半精度、单精度（TF32）和双精度等浮点数据类型的计算，使用 Tensor Core 执行特定数据类型的 GEMM 运算相较于使用 CUDA Core 可以提供 10 倍以上的性能提升。NVIDIA 在 CUDA 中提供了一组 WMMA API 用于开发者直接调用 Tensor Core 硬件单元执行矩阵乘法累加操作。但为了更灵活地使用 Tensor Core，本文采用内嵌 PTX 指令的形式调用 Tensor Core。图 1 展示了 PTX 中双精度 MMA 指令的计算布局。

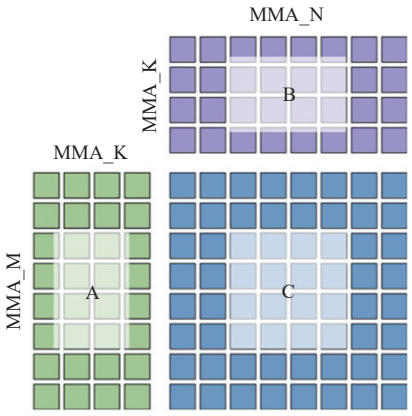



图 1 双精度 mma_m8n8k4 指令的计算布局

值得注意的是，尽管 Tensor Core 的设计初衷是优化稠密矩阵乘法，一些非 GEMM 的基本算法也能借助 Tensor Core 优异的计算能力获得性能的提高，如规约和扫描操作[17]、Stencil 计算[18]、SpMV[16]、稀疏矩阵–稠密矩阵乘法（SpMM）[19]和稀疏矩阵–矩阵乘法（SpGEMM）[20]等。因此，利用 Tensor Core 加速 CG 和 BiCGSTAB 方法中的关键计算模块，从而显著提升这两种方法的整体效率，具有很大的潜力。

## 2 算法实现

### 2.1 概述

Tensor Core 加速的 CG 和 BiCGSTAB 方法主要由 Host 端的数据准备和 Device 端的迭代步骤组成。下面将以 CG 方法为例介绍整个算法的流程，图 2 展示了一个使用 Tensor Core 加速关键计算组件的 CG 方法流程图。

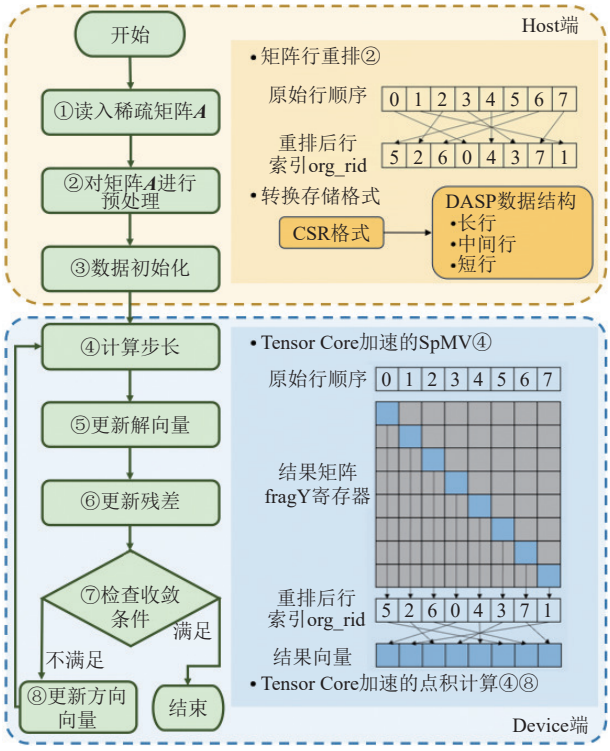



图 2 使用 Tensor Core 加速的 CG 方法的流程图

对于 Tensor Core 上的 CG 方法，在 Host 端进行数据准备时，首先需要对读入的稀疏矩阵 $\boldsymbol{A}$ 进行预处理，以满足 Tensor Core 对于输入输出矩阵的要求，从而执行后续在 Device 端的 SpMV 操作。Host 端的预处理工作包括矩阵行重排、存储格式转换，以及生成重排后行索引数组 org_rid。通过这些操作，稀疏矩阵被转化为适合使用 Tensor Core 计算的格式，同时保证了结果顺序的正确性。在 Device 端，CG 方法以迭代的形式计算步长、更新解向量、残差和方向向量。在每次迭代中，计算步长部分会先执行 Tensor Core 加速的

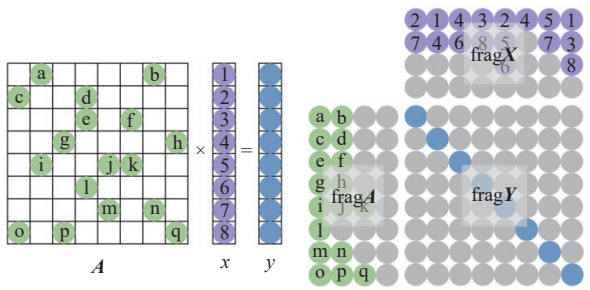
SpMV 操作完成计算，然后再使用 Tensor Core 加速的点积操作得到当前阶段的步长。随后，Device 端依次更新解向量、残差和方向向量。在更新方向向量阶段再次涉及点积计算，同样调用 Tensor Core 上的点积算法完成这部分的计算。

通过使用 Tensor Core 加速 CG 方法的关键组件 SpMV 和点积操作，实现 CG 方法对 Tensor Core 的高效利用。其中，读入稀疏矩阵、对读入矩阵进行预处理操作以及求解数据的初始化工作在 Host 端执行，计算步长、更新解向量和更新残差等迭代步骤在 Device 端执行。

与 CG 方法类似，BiCGSTAB 的计算过程也通过调用 Tensor Core 加速的 SpMV 和点积算法完成。与 CG 不同的是，BiCGSTAB 中包含了 2 次对 SpMV 算法的调用和 4 次对点积计算的调用。由于矩阵在预处理过程中同样地进行了行重排，SpMV 的结果向量顺序也对应于重排前的行顺序，因此在计算完成后，BiCGSTAB 同样会根据重排后行索引数组将结果向量恢复到原始行顺序，确保后续计算和最终结果的准确性。

### 2.2　Tensor Core 加速的 SpMV 算法的优化

SpMV 操作是一个稀疏矩阵$\boldsymbol{A}$与一个稠密向量$\boldsymbol{x}$相乘得到一个稠密向量$\boldsymbol{y}$。目前已有的 Tensor Core 上的 SpMV 方法 DASP[16] 设计了一种基于稀疏矩阵行长度分类的高效数据格式，以适配 Tensor Core 对输入输出数据的规格要求。在该方法中，矩阵的行根据非零元数量分为长行、中间行和短行 3 类，每一类行对应着不同的数据存储方式，也对应着采用不同的计算策略。图 3 展示了一个 8×8 大小的稀疏矩阵使用 Tensor Core 提供的尺寸为 m8n8k4 的 MMA 指令计算 SpMV 操作的示例。本文根据 CG 和 BiCGSTAB 的算法特性，对现有的使用 Tensor Core 加速的 SpMV 算法进行优化，使其能够更高效地处理这两种算法中常见的稀疏矩阵模式。




a. 8×8大小的矩阵$\boldsymbol{A}$的SpMV示例　　b. 利用MMA指令计算8×8的SpMV

图 3　利用 Tensor Core 计算一个 8×8 大小的稀疏矩阵的 SpMV 操作示例

在 CG 和 BiCGSTAB 方法中，Host 端主要是对读入的矩阵进行预处理操作，将其整理为使用 Tensor Core 进行矩阵计算所需要的形式。预处理主要包含了两部分：1）对矩阵的行进行重排，按照行中非零元的数量将矩阵行划分为长行、中间行和短行；2）将重排后的矩阵转换为使用 Tensor Core 计算 SpMV 所需的数据结构。然而，矩阵的重排会改变行的顺序，影响原始矩阵的特征并导致结果顺序错乱。因此，为了保证矩阵的原始特征，并在计算完成后将结果恢复为正确的顺序，本文在预处理过程中引入了一个重排后行索引数组 org_rid。该索引数组记录了重排后的行索引与矩阵原始行索引之间的映射关系，用于在 Device 端每次 SpMV 计算结束后，将有效结果按照原始行顺序写回到结果向量中。

在 Device 端计算步长 $a_j$ 这一步骤中，本文调用 Tensor Core 上的 SpMV 算法对已使用新格式存储的矩阵$\boldsymbol{A}$和方向向量$\boldsymbol{p}_j$进行 SpMV 计算，得到结果向量$\boldsymbol{Ap}_j$。在 SpMV 的计算阶段，通过将矩阵按行长度分为长行、中间行和短行，分别使用不同的存储和计算策略，充分发挥 Tensor Core 的性能来加速 SpMV 操作。由于矩阵在 Host 端已经进行了行重排，SpMV 的结果向量$\boldsymbol{Ap}_j$与重排后的矩阵行顺序对应。为了保证后续计算的正确性，Device 端需要根据 Host 端生成的重排后行索引数组 org_rid，将结果向量$\boldsymbol{Ap}_j$按照矩阵原始行的顺序写回。调用 Tensor Core 上的 SpMV 算法并结合重排后行索引数组完成结果向量的正确写回，既保证了计算的高效性，又确保了结果的准确性。

### 2.3　Tensor Core 加速的点积计算的优化

点积计算是线性代数中的一种基本运算，用于计算两个向量对应元素乘积的累加和，其本质是一个规约（Reduction）问题。文献 [17] 将规约问题转化为矩阵乘法，利用 Tensor Core的高吞吐量矩阵运算单元分块进行归约，最终实现高效的规约计算。基于这一思路，本文设计了一个使用 Tensor Core 加速的点积计算。具体而言，首先将相乘后的结果向量组织成适配 Tensor Core 的小矩阵块，然后通过 Tensor Core 的矩阵乘法完成局部积的累加归约，并最终将结果规约为标量。

图 4 以矩阵乘法指令 m4n4k2 为例，展示了使用 Tensor Core 进行点积运算的完整流程。首先，对两个输入向量中某一段的对应数据逐一相乘，将得到的结果直接存入矩阵$\boldsymbol{B}_i$中，由于 FP64 的计算

指令操作数并非方阵，这里参与计算的矩阵$\boldsymbol{A}$和$\boldsymbol{B}_i$都由两个子矩阵拼凑得到；然后，循环调用MMA指令对第一行为1其余行全部为0的立即矩阵（不需要加载数据，直接赋值的矩阵）$\boldsymbol{A}$与第一步得到的矩阵$\boldsymbol{B}_i$进行矩阵乘，结果累加给矩阵$\boldsymbol{C}_i$，这里的矩阵乘需要调用两次MMA指令完成；最后得到的$\boldsymbol{C}_N$将作为最后一次矩阵乘的左操作数与立即矩阵$\boldsymbol{A}^{\mathrm{T}}$进行矩阵乘法计算，其计算结果累加到一个全置0的立即矩阵$\boldsymbol{Z}$上，最终得到的结果矩阵$\boldsymbol{R}$中的第一个值$R(0,0)$即为该片段的和；最后，将各个片段的和累加给最终的输出值，为了避免读写冲突，该操作调用CUDA中的原子加操作完成。在实际代码中调用的是尺寸为m8n8k4的MMA指令，同时设置了一个线程块有4个Warp，一个片段的求和过程由一个Warp完成，一个片段的长度为256，这些参数都可以进行灵活调整。

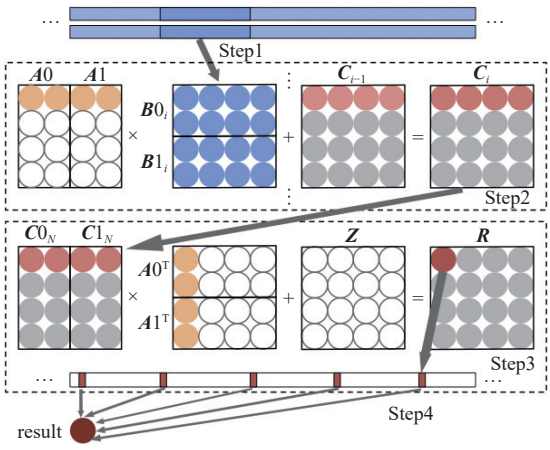



图4 使用Tensor Core加速点积计算的流程示例

## 3 实验结果

### 3.1 实验设置

本工作将前文提到的方法与CG和BiCGSTAB方法的基准实现在NVIDIA的A100 GPU（Ampere架构，驱动版本525.85.12，CUDA版本12.1）和H100 GPU（Hopper架构，驱动版本，CUDA版本12.1）分别进行了对比测试。CG和BiCGSTAB的基准实现分别调用了cuSPARSE v12.1数学库中的cusparseSpMV()函数和cuBLAS v12.1数学库中的cublasDdot()函数来实现CG和BiCGSTAB方法中SpMV操作和点积操作的计算。

为了验证本文提出的方法能够提升CG和BiCGSTAB方法的计算效率，本文没有通过判断残差控制迭代，而是设置了每种方法强制迭代100次进而收集计算的时间开销。在具体的测试中，每个矩阵对应的右手边向量设定为输入矩阵与全1向量的乘积，求解向量的初始值设为0（即不使用预条件）。同时，所有的数值都使用双精度浮点数类型（FP64）存储。

本文选取了SuiteSparse稀疏矩阵集[15]中的9个对称正定矩阵和9个非对称正定矩阵分别作为CG和BiCGSTAB方法的实验测试集。这18个矩阵覆盖了来自结构分析、流体力学、电路仿真和网络分析等多个实际应用领域，规模从小型到超大型尺寸不等，非零元分布和稀疏性差异显著，能够较全面地测试CG和BiCGSTAB方法在不同场景下的适用性和性能。表1展示了这18个代表性矩阵的特征。

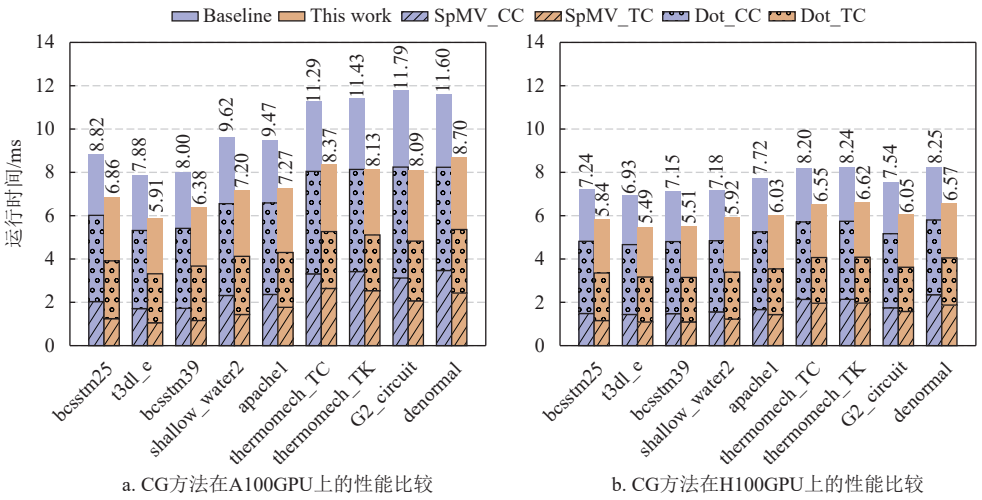
**表1 测试矩阵集**

| 对称正定矩阵 | | | 非对称正定矩阵 | | |
|---|---|---|---|---|---|
| 矩阵名称 | 阶数 | 非零元数 | 矩阵名称 | 阶数 | 非零元数 |
| bcsstm25 | 15 439 | 15 439 | dtoc | 24 993 | 69 972 |
| t3dl_e | 20 360 | 20 360 | shar_te2-b3 | 200 200 | 800 800 |
| bcsstm39 | 46 772 | 46 772 | NotreDame_www | 325 729 | 929 849 |
| shallow_water2 | 81 920 | 327 680 | cont-300 | 180 895 | 988 195 |
| apache1 | 80 800 | 542 184 | vfem | 93 476 | 1 434 636 |
| thermomech_TC | 102 158 | 711 558 | mc2depi | 525 825 | 2 100 225 |
| thermomech_TK | 102 158 | 711 558 | cop20k_A | 121 192 | 2 624 331 |
| G2_circuit | 150 102 | 726 674 | belgium_osm | 1 441 295 | 3 099 940 |
| denormal | 89 400 | 1 156 224 | debr | 1 048 576 | 4 194 298 |

### 3.2 与CUDA官方库的性能比较

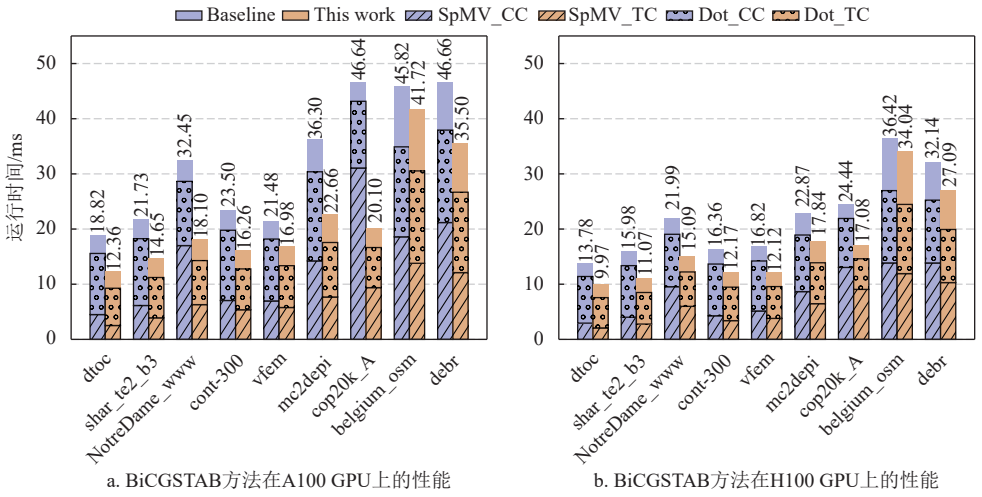
在NVIDIA A100和H100两个GPU上，分别将前文提到的Tensor Core加速的CG方法和BiCGSTAB方法与使用CUDA Core计算的基准实现进行了对比，记录了9个对称正定矩阵（使用CG方法求解）和9个非对称正定矩阵（使用BiCGSTAB方法求解）的device端总体求解时间、执行SpMV操作的时间开销以及执行点积的时

间开销。图 5 和图 6 展示了 CG 方法和 BiCGSTAB 方法的测试结果。




a. CG方法在A100GPU上的性能比较　　b. CG方法在H100GPU上的性能比较

图 5　CG 方法在 A100 和 H00 GPU 上的性能比较




a. BiCGSTAB方法在A100 GPU上的性能　　b. BiCGSTAB方法在H100 GPU上的性能

图 6　BiCGSTAB 方法在 A100 和 H00 GPU 上的性能比较

对于 CG 方法，与调用 cuSPARSE 数学库的基准实现版本相比，本文提出的 Tensor Core 加速的 CG 方法分别在 NVIDIA A100 和 H100 GPU 上实现了平均 1.34 倍和 1.25 倍（最高 1.46 倍和 1.30 倍）的加速。对于 CG 方法中 SpMV 操作的开销部分，本工作采用的算法相比调用 cuSPARSE 数学库的 SpMV 方法在两个 GPU 上分别获得了 1.47 倍和 1.21 倍（最高 1.63 和 1.34 倍）的加速。对于点积计算部分，使用 Tensor Core 计算的点积相比调用 cuBLAS 数学库的点积分别有 1.66 倍和 1.62 倍（最高 1.86 和 1.70 倍）的加速。

对于 BiCGSTAB 方法，从 Device 端的总体运行时间来看，与调用 cuSPARSE 数学库的基准实现版本相比，本工作实现的 Tensor Core 加速的 BiCGSTAB 方法分别在 NVIDIA A100 和 H100 GPU 上实现了平均 1.54 倍和 1.33 倍（最高 2.32 倍和 1.46 倍）的加速。对于 SpMV 操作的开销部分，本工作采用的算法相比调用 cuSPARSE 数学库的实现在两个 GPU 上分别达到了 1.87 倍和 1.37 倍（最高 3.31 倍和 1.59 倍）的加速。对于点积计算部分，本工作实现的点积算法相比调用 cuBLAS 数学库的点积分别有 1.49 倍和 1.45 倍（最高 1.72 倍和 1.65 倍）的加速。

以矩阵 shallow_water2 为例，进一步对 CG 方法的测试结果进行分析。通过观察该矩阵的结构可以发现，矩阵 shallow_water2 的非零元分布比较规则，每行的长度都为 4，与本文在计算 SpMV 时调用的 MMA 指令的 $K$ 维度的大小刚好一致。在求解该矩阵的测试中测得，本文提出的 CG 方法的性能在 A100 和 H100 GPU 上是基准实现版本的

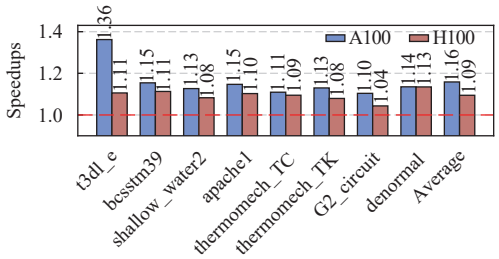
1.34 倍和 1.21 倍，其中 SpMV 的性能分别是调用 cuSPARSE 数学库的 1.61 倍和 1.27 倍，点积计算的性能分别是调用 cuBLAS 数学库的 1.58 倍和 1.51 倍。同样地，以矩阵 cop20k_A 为例来进一步分析 BiCGSTAB 的测试结果。该矩阵每行非零元数量比较均匀，几乎所有的行在 SpMV 阶段都被划分到了中间行类别中。对于该矩阵，本文提出的 Tensor Core 加速的 BiCGSTAB 方法的总体性能是基准实现的 2.32 倍和 1.43 倍，其中 SpMV 部分是 cuSPARSE 数学库的 3.31 倍和 1.44 倍，而点积计算的性能分别是调用 cuBLAS 数学库的 1.66 倍和 1.60 倍。

除此之外，由于 BiCGSTAB 方法的每次迭代要执行 2 次 SpMV 和 4 次点积操作，整个算法中 SpMV 和点积的总占比较高，因此 BiCGSTAB 方法的加速效果比 CG 方法的加速效果更明显。

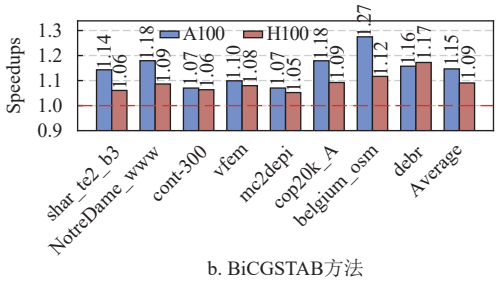
### 3.3 Tensor Core 的有效性分析

为了验证使用 Tensor Core 计算 SpMV 和点积时能够为 CG 和 BiCGSTAB 方法带来的性能提升，本文进一步设计了有效性对比实验。具体来说，将本文提出的算法中调用 Tensor Core 进行 MMA 计算的部分替换为使用 CUDA Core 实现的 MMA 计算，保持其他部分完全一致，从而评估两种硬件单元对整体算法的影响。

图 7 展示了两种方法的性能对比结果，直观反映了使用 Tensor Core 加速的优越性。对于 CG 方法，Tensor Core 计算 MMA 版本相较使用 CUDA Core 计算 MMA 版本在 A100 和 H100 上的平均加速比分别为 1.16 和 1.09，其中性能提升最显著的是矩阵 t3dl_e，在 A100 上达到了 1.36 的加速比。对于 BiCGSTAB 方法，与 CUDA Core 计算 MMA 版本相比，Tensor Core 版本在 A100 和 H100 两款 GPU 上的平均加速比分别为 1.15 和 1.09，其中矩阵 cop20k_A 在 A100 上的加速比高达 1.27。这些结果表明，使用 Tensor Core 能显著加速 CG 和 BiCGSTAB 的整体求解过程。




a. CG方法




b. BiCGSTAB方法

图 7 CG 和 BiCGSTAB 方法使用 Tensor Core 和 CUDA Core 计算 MMA 操作的性能比较

## 4 结束语

本文提出了一个使用 Tensor Core 加速的 CG 和 BiCGSTAB 算法。使用 Tensor Core 加速 CG 和 BiCGSTAB 方法中的关键计算模块 SpMV 和点积操作，以充分发挥 Tensor Core 的计算优势，从而提升 CG 和 BiCGSTAB 方法的整体性能。实验证明，本文提出的方法在两款 NVIDIA GPU 上都展现出了显著的性能提升。

## 参 考 文 献

[1] HESTENES M R, STIEFEL E. Methods of conjugate gradients for solving linear systems[J]. Journal of Research of the National Bureau of Standards, 1952, 49(6): 409-436.

[2] VORST H A. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems[J]. SIAM Journal on Scientific and Statistical Computing, 1992, 13(2): 631-644.

[3] 李亿渊, 薛巍, 陈德训, 等. 稀疏矩阵向量乘法在申威众核架构上的性能优化[J]. 计算机学报, 2020, 43(6): 1010-1024.
LI Y Y, XUE W, CHEN D X, et al. Performance optimization for sparse matrix-vector multiplication on sunway architecture[J]. Chinese Journal of Computers, 2020, 43(6): 1010-1024.

[4] 吴洋, 赵永华, 纪国良. 一类大规模稀疏矩阵特征问题求解的并行算法[J]. 数值计算与计算机应用, 2013, 34(2): 136-146.
WU Y, ZHAO Y H, JI G L. Parallel solving large-scale sparse matrix eigenvalue problem[J]. Journal on Numerical Methods and Computer Applications, 2013, 34(2): 136-146.

[5] 李佳佳, 张秀霞, 谭光明, 等. 选择稀疏矩阵乘法最优存储格式的研究[J]. 计算机研究与发展, 2014, 51(4): 882-894.
LI J J, ZHANG X X, TAN G M, et al. Study of choosing the optimal storage format of sparse matrix vector multiplication[J]. Journal of Computer Research and Development, 2014, 51(4): 882-894.

[6] 杜臻, 谭光明. 稀疏矩阵向量乘的自动调优[J]. 计算物理, 2024, 41(1): 33-39.
DU Z, TAN G M. Auto-tuning for sparse matrix-vector multiplication[J]. Chinese Journal of Computational

Physics, 2024, 41(1): 33-39.
[7] 徐小文. 并行代数多重网格算法: 大规模计算应用现状与挑战[J]. 数值计算与计算机应用, 2019, 40(4): 243-260.
XU X W. Parallel algebraic multigrid methods: State-of-the art and challenges for extreme-scale applications[J]. Journal on Numerical Methods and Computer Applications, 2019, 40(4): 243-260.
[8] 毛润彰, 杜皓, 田鸿运, 等. 几类典型应用的代数多重网格算法并行可扩展瓶颈分析[J]. 计算物理, 2024, 41(4): 403-417.
MAO R Z, DU H, TIAN H Y, et al. Analysis of parallel scalability bottleneck for algebraic multigrid in typical real applications[J]. Chinese Journal of Computational Physics, 2024, 41(4): 403-417.
[9] 迟利华, 刘杰, 李晓梅. 稀疏近似逆预条件子及其并行计算[J]. 计算机学报, 2000, 23(3): 255-260.
CHI L H, LIU J, LI X M. Parallel sparse approximate inverse preconditioners[J]. Chinese Journal of Computers, 2000, 23(3): 255-260.
[10] GRIGORI L, TISSOT O. Scalable linear solvers based on enlarged Krylov subspaces with dynamic reduction of search directions[J]. SIAM Journal on Scientific Computing, 2019, 41(5): C522-C547.
[11] YAMAZAKI I, CARSON E, KELLEY B. Mixed precision s-step conjugate gradient with residual replacement on GPUs[C]//Proceedings of the IEEE International Parallel and Distributed Processing Symposium. New York: IEEE, 2022: 886-896.
[12] BURGESS J. RTX on–The NVIDIA turing GPU[C]// Proceedings of the IEEE Hot Chips 31 Symposium. New York: IEEE, 2019: 1-27.
[13] CHOQUETTE J, GANDHI W. NVIDIA A100 GPU: Performance & innovation for GPU computing[C]// Proceedings of the IEEE Hot Chips 32 Symposium. New York: IEEE, 2020: 1-43.
[14] CHOQUETTE J. Nvidia hopper GPU: Scaling performance[C]//Proceedings of the IEEE Hot Chips 34 Symposium. New York: IEEE, 2022: 1-46.
[15] DAVIS T A, HU Y F. The university of Florida sparse matrix collection[J]. ACM Transactions on Mathematical Software, 2011, 38(1): 1-25.
[16] LU Y C, LIU W F. DASP: Specific dense matrix multiply-accumulate units accelerated general sparse matrix-vector multiplication[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2023: 1-14.
[17] DAKKAK A, LI C, XIONG J J, et al. Accelerating reduction and scan using tensor core units[C]//Proceedings of the ACM International Conference on Supercomputing. New York: ACM, 2019: 46-57.
[18] CHEN Y T, LI K, WANG Y H, et al. ConvStencil: Transform stencil computation to matrix multiplication on tensor cores[C]//Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming. New York: ACM, 2024: 333-347.
[19] OKANOVIC P, KWASNIEWSKI G, LABINI P S, et al. High performance unstructured SpMM computation using tensor cores[C]//Proceedings of the SC24: International Conference for High Performance Computing, Networking, Storage and Analysis. New York: IEEE, 2024: 1-14.
[20] LU Y C, ZENG L J, WANG T C, et al. AmgT: Algebraic multigrid solver on tensor cores[C]//Proceedings of the SC24: International Conference for High Performance Computing, Networking, Storage and Analysis. New York: IEEE, 2024: 1-16.

责任编辑　张　莉