arXiv:2304.04612v1 [cs.DC] 10 Apr 2023

TensorCore 上用于 RandNLA 的混合精度随机投影

大友弘之[1], † 横田理央[1], ††

[1] 东京工业大学
† ootomo.h@rio.gsic.titech.ac.jp
†† rioyokota@gsic.titech.ac.jp

摘要

随机投影能够在捕捉数据结构的同时降低数据维度，是机器学习、信号处理和信息检索中的基础工具；如今，这些领域都需要处理大量数据。RandNLA（随机数值线性代数）利用随机投影降低张量低秩分解的计算复杂度，并求解最小二乘问题。虽然随机投影的计算只涉及简单的矩阵乘法，但其渐近计算复杂度通常高于 RandNLA 算法中的其他运算。因此，已有多项研究提出了降低其计算复杂度的方法。我们提出了一种面向单精度张量、在 NVIDIA GPU 上利用张量核心实现的快速混合精度随机投影方法。我们利用随机矩阵对精度要求较低这一特点，在张量核心上开发了高度优化的 FP32 与 FP16 矩阵乘法——SHGEMM（单精度与半精度 GEMM），其中随机矩阵以 FP16 格式存储。与基线单精度实现相比，我们的方法在保持精度的同时，可将随机化奇异值分解的计算速度提升 1.28 倍，将随机投影高阶奇异值分解的计算速度提升 1.75 倍。

1 引言

随机投影是一种稳健的数据降维与压缩工具，它通过将数据投影到低维子空间，在压缩数据的同时保留其结构。它广泛应用于机器学习 [8, 42, 17, 7, 18], 计算机视觉 [24, 41, 29], 数据库搜索 [2, 21], 以及其他应用领域 [33, 46]。随着我们处理的此类压缩格式数据量不断增加，随机投影的性能变得至关重要。RandNLA（随机数值线性代数） [12]是一类数值算法，利用随机投影及其他随机化方法，降低张量低秩近似 [20, 39, 15, 31, 3, 6],和最小二乘方法 [38]的计算复杂度。例如，随机化奇异值分解（SVD）是一种快速的矩阵低秩近似算法，其近似秩预先给定 [20]。虽然使用奇异值分解对矩阵进行低秩近似是一项基础操作，但奇异值分解的计算复杂度很高。随机化奇异值分解及其变体降低了计算复杂度，应用于图像与数据压缩 [14], 矩阵补全 [16], 数字水印 [5, 40], 以及其他研究领域 [44, 45, 25, 43, 22]。例如，可以通过 scikit-learn Python 软件包使用这些方法[37], 。尽管随机投影只是输入矩阵与随机矩阵的一次简单乘法，但在 RandNLA 算法中，其渐近计算复杂度高于其他部分，例如对小型瘦高矩阵进行奇异值分解和 QR分解。因此，许多研究致力于利用结构化随机矩阵 [20], 稀疏随机矩阵 [2, 26], 和哈达玛矩阵 [30]来降低随机投影的计算复杂度。然而，据我们所知，利用最新硬件特性加速随机投影的研究仍然寥寥无几。

为满足深度学习对快速矩阵乘法计算的巨大需求，人们开发了多种专用于矩阵乘法的计算单元，例如 NVIDIA Tensor Core [10], AMD Matrix Core [4], Google TPU [23], 以及 Preferred Networks MN-Core [27]。NVIDIA Tensor Core 是一种用于混合精度矩阵乘法与加法的计算单元，在 A100 GPU 上的理论峰值性能为每秒 312 万亿次浮点运算。虽然用于乘法的两个矩阵以低精度（FP16 或 TF32）存储，但内部计算以 FP32 执行，因此可以得到 FP32 格式的结果矩阵。然而，由于需要将输入 FP32 矩阵转换为低精度，在张量核心上计算单精度矩阵乘法时，结果精度会下降。为恢复精度，Markidis 等人 提出了一种利用补偿求和恢复精度的方法。该方法将每个输入 FP32 矩阵拆分为两个 FP16 矩阵之和，并在张量核心上对各个拆分矩阵分别相乘后求和 [28]。尽管从理论上讲，该方法能够恢复精度，但实践中的精度仍低于单精度 [34]。我们的先前工作表明，张量核心内部使用的舍入模式（RZ）是精度下降的原因。我们提出了 TCEC-SGEMM（带误差校正的张量核心）方法，通过在 FP32 SIMT 核心上执行累加，避免在张量核心内部进行累加。我们已证明，在精度处于同一水平的同时，该方法在 A100 GPU 上的性能超过了 FP32 SIMT 核心的理论峰值性能。作为在张量核心上计算高精度 GEMM 的另一种拆分方法，Mukunoki 等人 采用 Ozaki 方案 [36] ，并证明该方法能够在张量核心上执行双精度矩阵乘法 [32]。Ozaki 方案与 TCEC-SGEMM 的主要区别在于：前者根据各行或各列共用的阈值拆分输入矩阵的尾数，并对拆分后的矩阵执行无误差矩阵乘法；后者则根据每个元素的局部阈值拆分尾数，且拆分后矩阵的所有乘法都会引入舍入误差。尽管 Mukunoki 等人证明，该方法在 NVIDIA TITAN RTX GPU 上执行与双精度等效的矩阵乘法时，速度能够超过 FP64 的理论峰值性能，但单精度乘法却较慢。

我们提出一种在张量核心上执行的混合精度随机投影方法，以提高单精度张量的随机投影吞吐量。在此前的研究中，随机投影所用高斯随机矩阵的数据类型与输入矩阵相同（FP32），因为这样可以利用针对各类处理器高度优化的 GEMM 实现。另一方面，我们在随机投影中使用 FP16 随机矩阵，因为我们提出的 GEMM 方法——**SHGEMM** （单精度与半精度 GEMM）——能够将其与 FP32 输入矩阵相乘，速度比 SGEMM 更快，同时保持精度。因此，使用低精度随机矩阵可以减少随机矩阵的内存使用量，并提高计算吞吐量。我们还表明，该方法在保持精度的同时，能够提升 RandNLA 算法的吞吐量。

本文的主要贡献概述如下。

- 我们研究了高斯随机矩阵低精度表示的性质，并表明这种表示可用于随机投影。

- 我们实现了 SHGEMM，这是一种使用 Tensor Cores 对 FP32 和 FP16 矩阵进行矩阵乘法的方法。我们展示其舍入误差分析、精度和吞吐量，并研究吞吐量瓶颈。

- 我们在两种 RandNLA 算法中评估了低精度高斯随机投影和 SHGEMM。与基线 FP32 实现相比，我们将随机化奇异值分解的吞吐量提升了 1.28 倍，将随机投影高阶奇异值分解的吞吐量提升了 1.49 倍，同时保持了精度。

2 背景

2.1 使用奇异值分解进行低秩近似

对于复矩阵 $\mathbf{A} \in \mathbb{C}^{m,n}$，奇异值分解（SVD）将 $\mathbf{A}$ 分解为三个矩阵的乘积：$\mathbf{A} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^{\top}$。矩阵 $\mathbf{U}$ 和 $\mathbf{V}$ 是酉矩阵，而 $\mathbf{\Sigma}$ 是一个对角矩阵，其对角元素 $\sigma_1, \cdots, \sigma_k$ 是 $\mathbf{A}$ 的奇异值，k 是 $\mathbf{A}$的秩。通过截断 $[p+1$，可以得到基于 SVD 的 $\mathbf{A}$ 的 p秩近似（截断奇异值分解；tSVD）：截取$\mathbf{U}$,$\mathbf{V}$ 的 $k]$ 个列向量以及 $\mathbf{\Sigma}$ 的对角元素，如下所示：

$$\mathbf{A} \sim \hat{\mathbf{A}} = \hat{\mathbf{U}} \cdot \mathbf{\Sigma}_1 \cdot \hat{\mathbf{V}^{\top}}$$

其中

$$\hat{\mathbf{U}} \leftarrow \mathbf{U}_{[1:p]}, \hat{\mathbf{V}} \leftarrow \mathbf{V}_{[1:p]}, \mathbf{\Sigma}_1 \leftarrow \operatorname{diag}(\sigma_1, \cdots, \sigma_p).$$

埃卡特界定了近似精度——杨氏定理 [13]。

定理 1（Eckart–Young定理）

$$||\mathbf{A} - \hat{\mathbf{A}}||_F = ||\mathbf{\Sigma}_2||_F, \tag{1}$$

其中 $\mathbf{\Sigma}_2 = diag(\sigma_{p+1}, \cdots, \sigma_k)$，而 $||\cdot||_F$表示 *Frobenius*范数。

由于 $m \times n$ 矩阵的奇异值分解计算复杂度为 $\mathcal{O}(mn \cdot \min(m,n))$，且计算量很大，因此当近似秩已知时，我们不会对输入矩阵计算完整的奇异值分解。相反，我们使用一种基于秩揭示QR分解的算法 [19]，以 $\mathcal{O}(mnp)$ 的计算量得到相同的近似结果。

当奇异值衰减缓慢时，我们可以使用幂迭代方案，将输入矩阵 ˆ变换为$\mathbf{A}$，计算如下：

$$\hat{\mathbf{A}} = \left(\mathbf{A}\mathbf{A}^{\top}\right)^q \mathbf{A} = \mathbf{U} \cdot \mathbf{\Sigma}^{2q+1} \cdot \mathbf{V}^{\top},$$

其中，q 是幂迭代的次数。

2.2 RandNLA 中的随机投影

本节以随机化奇异值分解为例，说明 RandNLA 中的随机投影方法。已知近似秩时，可以使用随机化奇异值分解算法。在随机化奇异值分解的p-秩近似中，我们按如下方式计算矩阵$\mathbf{Y} \in \mathbb{R}^{m \times p}$：

$$\mathbf{Y} = \mathbf{A}\mathbf{\Omega} \tag{2}$$

其中，$\mathbf{\Omega} \in \mathbb{R}^{n \times p}$ 是一个随机矩阵。由于$\mathbf{Y}$ 的列向量是$\mathbf{A}$的列向量的线性组合，因此这两个矩阵共享标准正交向量。因此，例如，对$\mathbf{Y}$进行QR分解得到的正交矩阵$\mathbf{Q}$，也是$\mathbf{A}$的标准正交向量。由此，$\mathbf{A}$ 可近似为：

$$\mathbf{A} \sim \mathbf{Q}\mathbf{Q}^{\top}\mathbf{A}. \tag{3}$$

然后，我们可以通过计算 $\mathbf{Q}^{\top}\mathbf{A}$ 的奇异值分解来计算 p-秩近似，如下所示：

$$\hat{\mathbf{U}}', \mathbf{\Sigma}_1, \hat{\mathbf{V}} \leftarrow \operatorname{SVD}\left(\mathbf{Q}^{\top}\mathbf{A}\right), \hat{\mathbf{U}} \leftarrow \mathbf{Q}\hat{\mathbf{U}}'.$$

式 (2) 的计算称为**随机投影**。在实践中，我们将 p 设为 ˆ$p = p + s$，其中$s(\geq 2)$ 是一个过采样参数。

算法 1随机化奇异值分解

要求： 输入矩阵 $\mathbf{A} \in \mathbb{C}^{m\times n}$, $\mathbf{\Omega} \in \mathbb{R}^{n\times}$ $\hat{p}$, 目标秩 $p \in \mathbb{N}$

确保： $\hat{\mathbf{U}}, \mathbf{\Sigma}_1, \hat{\mathbf{V}}$ $s.t.\ \hat{\mathbf{A}} = \hat{\mathbf{U}} \cdot \mathbf{\Sigma}_1 \cdot \hat{\mathbf{V}}^\top$

1: $\mathbf{Y} \leftarrow \mathbf{A} \cdot \mathbf{\Omega}$

2: $\mathbf{Q}$,$\mathbf{R} \leftarrow$ QR分解($\mathbf{Y}$)

3: $\mathbf{B} \leftarrow \mathbf{Q}^\top \cdot \mathbf{A}$

4: $\hat{\mathbf{U}}'$, $\mathbf{\Sigma}_1$, $\hat{\mathbf{V}} \leftarrow$ t奇异值分解($\mathbf{B}$ $rank = p$)

5: $\hat{\mathbf{U}} \leftarrow \mathbf{Q} \cdot \hat{\mathbf{U}'}$

当使用**高斯随机矩阵 G**，且其中每个元素均由 $\mathcal{N}(0, 1)$ 生成时，随机投影的精度界限如下 [20]:

$$\mathbb{E}_{\mathbf{G}} ||\mathbf{A} - \mathbf{Q}\mathbf{Q}^\top \mathbf{A}||_F \leq \left(\sqrt{1 + p/(s-1)}\right) ||\mathbf{\Sigma}_2||_F, \tag{4}$$

其中 $\mathbb{E}_X$ 是关于 X 的期望，$||\cdot||_F$ 是 Frobenius范数。

我们在算法 1 中展示了随机化奇异值分解的算法。在该算法中，QR分解（第 2 行）和奇异值分解（第 4 行）的渐近计算复杂度分别为 $\mathcal{O}(mp^2)$ 和 $\mathcal{O}(np^2)$，而随机投影（第 1 行）的计算复杂度为 $\mathcal{O}(mnp)$。因此，由于 $p \ll m$, n 通常，随机投影是算法中计算开销最大的部分，随机化奇异值分解的总渐近复杂度为 $\mathcal{O}(mnp)$。虽然这与基于秩揭示QR分解的 tSVD 算法的复杂度相同，但已有实验表明，随机化奇异值分解比该算法更快 [20]。此外，为了降低随机投影的复杂度，一些研究尝试使用随机矩阵来替代高斯随机矩阵。例如，使用子采样随机傅里叶变换时，计算复杂度为 $\mathcal{O}(mn \cdot \log(p))$ [20]。我们也可以使用稀疏随机矩阵进行随机投影，尽管它们最初并非在 RandNLA 的背景下提出。这些矩阵从非零元素的角度以及元素值的选取上来看都是稀疏的。例如，Achlioptas 提出了一种随机矩阵$\mathbf{\Omega} \in \mathbb{R}^{n\times q}$，其中（$i$, j）元素按如下方式确定 [2]:

$$\mathbf{\Omega}_{(i,j)} = \sqrt{s} \times \begin{cases} +1 & \text{with probability } 1/2\text{s} \\ 0 & \text{with probability } 1 - 1/\text{s} \\ -1 & \text{with probability } 1/2\text{s} \end{cases} \tag{5}$$

对于 $s = 1$ 和 3. Li等人提出了一种极稀疏随机投影，其中 $s \gg 3$，特别是 $s = \sqrt{n}$ 或$n/\log\ n$ [26]。

2.3 张量核心上的单精度 GEMM 仿真

NVIDIA Tensor Core 是用于在 NVIDIA GPU 上进行固定大小矩阵乘法和加法的混合精度计算单元。在 Tensor Core 上计算大型矩阵乘法时，我们会拆分输入矩阵，并将得到的矩阵求和。输入 Tensor Core 的矩阵数据类型仅限于低精度类型，如 FP16、TF32 等。尽管乘法和加法以 FP32 等高精度执行，如图 1 所示，但由于输入矩阵必须转换为低精度类型，因此在 Tensor Core 上计算单精度 GEMM 时，精度会下降。我们的先前工作（TCEC-SGEMM 方法）通过补偿求和以及避免在 Tensor Core 内部进行舍入（RZ，即向零舍入） [34]，恢复了这一精度。我们已证明，该方法的性能可超过 FP32 SIMT Core 的峰值性能，同时精度达到单精度水平。在我们的方法中，单精度矩阵乘法 $\mathbf{C}_{\mathrm{F32}} \leftarrow \mathbf{A}_{\mathrm{F32}} \cdot \mathbf{B}_{\mathrm{F32}}$

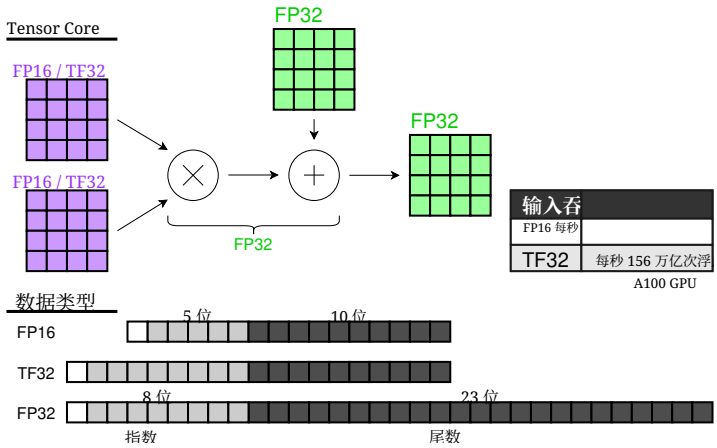


图1：我们使用的张量核心的输入、输出和计算精度。

近似计算如下：

$$\mathbf{A}_{\text{low}} \leftarrow \text{toLow}\left(\mathbf{A}_{\text{F32}}\right) \tag{6}$$

$$\Delta\mathbf{A}_{\text{low}} \leftarrow \text{toLow}\left(\left(\mathbf{A}_{\text{F32}} - \text{toF32}\left(\mathbf{A}_{\text{low}}\right)\right) \times 2^{11}\right) \tag{7}$$

$$\mathbf{B}_{\text{low}} \leftarrow \text{toLow}\left(\mathbf{B}_{\text{F32}}\right) \tag{8}$$

$$\Delta\mathbf{B}_{\text{low}} \leftarrow \text{toLow}\left(\left(\mathbf{B}_{\text{F32}} - \text{toF32}\left(\mathbf{B}_{\text{low}}\right)\right) \times 2^{11}\right) \tag{9}$$

$$\mathbf{C}_{\text{F32}} \sim \mathbf{A}_{\text{low}} \cdot \mathbf{B}_{\text{low}} + \left(\Delta\mathbf{A}_{\text{low}} \cdot \mathbf{B}_{\text{low}} + \mathbf{A}_{\text{low}} \cdot \Delta\mathbf{B}_{\text{low}}\right) \times 2^{-11}, \tag{10}$$

其中，toLow 将 FP32 矩阵转换为低精度格式，即 FP16 或 TF32；toF32 则将低精度矩阵转换为 FP32。由于公式（10）中的矩阵乘法是在张量核心上以高精度计算的，因此该方法能够以单精度计算矩阵乘法。此外，我们在 FP32 SIMT 核心上计算公式（10）中 $\mathbf{A}_{\text{low}} \cdot \mathbf{B}_{\text{low}}$ 部分加法，以便在舍入时使用舍入到最近值（RN）。我们将处理 FP16 矩阵的张量核心称为“FP16 张量核心”，将处理 TF32 矩阵的张量核心称为“TF32 张量核心”。

3 通过低精度高斯随机矩阵进行随机投影

3.1 高斯随机值的浮点表示

Halko 等人的随机投影理论误差界（式 (4)）是在隐含假设高斯随机值 $g \in \mathcal{N}(0,\ 1)$具有无限精度的前提下通过数学方法证明的。另一方面，我们将 g 表示为浮点值格式 eXmY，其中指数占 X 位，尾数占 Y 位，并记为 g_{eXmY}。例如，FP32 为 e8m23，FP16 为 e5m10。此外，我们使用 RN 进行舍入，这也是许多现代处理器（包括 NVIDIA GPU）的默认舍入方式。由于舍入误差， g 与 g_{eXmY} 之间的关系如下。

$$g_{\text{eXmY}} = g + \Delta g \tag{11}$$

其中 Δg 是舍入误差。在 RN 的情况下，满足以下不等式。

$$-\exp_\text{of}(g_{\text{eXmY}}) \times u_Y \leq \Delta g \leq \exp_\text{of}(g_{\text{eXmY}}) \times u_Y, \tag{12}$$

其中 exp of(·) 是浮点值的指数，且 $u_Y = 2^{-(Y+1)}$。

格式	$p^{\text{of}}_{\texttt{eXmY}}$	$p^{\text{未规格化}}_{\texttt{eXmY}}$	$p^{\text{uf}}_{\texttt{eXmY}}$
FP8 1 (e4m3)	$\ll 2-2\Phi(2^3)$ $< 1\times10^{-12}$	6×10^{-3}	8×10^{-4}
FP8 2 (e5m2)		2×10^{-5}	6×10^{-6}
FP16 (e5m10)		2×10^{-5}	2×10^{-8}
bfloat (e8m7)		$\ll 2\Phi(2^{-4^5})-1$ $< 2\times10^{-12}$	
TF32 (e8m10)			
FP32 (e8m23)			

	$N^{1\sigma}_{\texttt{eXmY}}$	$N^{2\sigma}_{\texttt{eXmY}}$	$N^{4\sigma}_{\texttt{eXmY}}$
FP8 1 (e4m3)	111	127	143
FP8 2 (e5m2)	119	127	135
FP16 (e5m10)	30,719	32,767	34,815
bfloat (e8m7)	32,511	32,767	33,023
TF32（e8m10）	260,095	262,143	264,191
FP32（e8m23）	2,130,706,431	2,147,483,647	2,164,260,863

表 1：以 eXmY 浮点数表示的高斯随机值的属性。**上方：** $p^{\text{of}}_{\text{eXmY}}$、$p^{\text{uf}}_{\text{eXmY}}$ 和 $p^{\text{not-normalized}}{}_{\text{eXmY}}$ 分别是上溢、下溢以及成为非规格化数或发生下溢的概率。**下方：** 在 σ、2σ 和4σ 范围内的浮点数数量。

3.2 低精度高斯随机值的性质

浮点数格式所表示的高斯随机值的性质因格式而异。我们考察以下三个性质。

1. 上溢和下溢概率。
2. 高斯分布中特定范围内的元素数量。
3. 均值与方差。

3.2.1 上溢和下溢概率

如果高斯随机值以不可忽略的概率发生上溢，我们就无法正确计算随机投影。另一方面，即使下溢以较高概率发生，仍然可以进行计算，因为这只是一个稀疏随机矩阵 [26]。我们从理论上研究这些概率。

可表示为 eXmY 的最大正值 max_{eXmY} 计算如下：

$$\text{max}_{\text{eXmY}} = 0b0'\underbrace{11\cdots10}_{\text{X bit}}'\underbrace{11\cdots1}_{\text{Y bit}} \tag{13}$$

$$= 2^{2^X-2-\text{bias}(X)}\times\left(1-2^{-(Y+1)}\right) \tag{14}$$

$$= 2^{2^{X-1}-1}\times\left(1-2^{-(Y+1)}\right), \tag{15}$$

其中，$\text{bias}(X) = 2^{X-1}-1$ 是 eXmY 的指数偏置。因此，溢出概率 $p^{\text{of}}_{\text{eXmY}}$，也就是说，出现大于 max_{eXmY} 的值的概率，计算如下：

$$p^{\text{of}}_{\text{eXmY}} = 2\times(1-\Phi(\text{max}_{\text{eXmY}})) \tag{16}$$

其中，$\Phi(\cdot)$ 是高斯分布 $\mathcal{N}(0,1)$的概率密度函数。另一方面，可表示为规格化的最小正值 $\text{min}^{\text{normalized}}_{\text{eXmY}}$ 和 $\text{min}^{\text{denormalized}}_{\text{eXmY}}$

eXmY 的规格化值和非规格化值分别按如下方式计算：

$$\begin{aligned}\min_{\text{eXmY}}^{\text{normalized}} &= 0b0'\underbrace{00\cdots01}_{\text{X bit}}'\underbrace{00\cdots0}_{\text{Y bit}} \\ &= 2^{1-\text{bias}(X)} = 2^{2-2^{X-1}}\end{aligned}$$

$$\begin{aligned}\min_{\text{eXmY}}^{\text{denormalized}} &= 0b0'\underbrace{00\cdots00}_{\text{X bit}}'\underbrace{00\cdots01}_{\text{Y bit}} \\ &= 2^{-\text{bias}(X)} \times 2^{-Y} = 2^{1-2^{X-1}} \times 2^{-Y}.\end{aligned}$$

因此，下溢概率 $p_{\text{eXmY}}^{\text{uf}}$ 和非规格化概率 $p^{\text{not-normalized}}{}_{\text{eXmY}}$ 计算如下：

$$\begin{aligned}p_{\text{eXmY}}^{\text{uf}} &= 2 \times (\Phi(\min_{\text{eXmY}}^{\text{denormalized}}) - 1/2) \\ p_{\text{eXmY}}^{\text{not-normalized}} &= 2 \times (\Phi(\min_{\text{eXmY}}^{\text{normalized}}) - 1/2).\end{aligned}$$

表 1 顶部列出了若干浮点格式下 $p_{\text{eXmY}}^{\text{of}}$，$p_{\text{eXmY}}^{\text{uf}}$ 和 $p^{\text{not-normalized}}{}_{\text{eXmY}}$ 的具体值。由于随机投影中随机矩阵的元素数量通常可达 $\sim 10^8$ ($n < 10^5$，$p < 10^3$)，当指数长度 X 大于 3 时，溢出概率可以忽略不计。

3.2.2 $2^s\sigma$ 范围内的元素数量

如果高斯分布某个特定范围内的元素数量太少，那么高斯随机矩阵可能不满秩，尽管这种情况发生的概率极低；式 (4) 的证明中假设了这一点。举一个极端例子，如果只有一个值可能出现，那么随机矩阵的秩为 0 或 1。我们研究高斯分布 $2^s\sigma$ 范围内的元素数量。

正规格化数元素 v_{eXmY} 在高斯分布 $\mathcal{N}(0, \sigma = 1)$ 的 $2^s\sigma$ 范围内满足以下不等式：

$$\min_{\text{eXmY}}^{\text{normalized}} \le v_{\text{eXmY}} < 0b0'\underbrace{**\cdots**}_{=s+\text{bias}(X)}'\underbrace{00\cdots0}_{\text{Y bit}}. \tag{17}$$

因此，$2^s\sigma$ 中包括 0在内的规格化数字元素数量 $N^{2^s}{}_{\text{eXmY}}$ 计算如下：

$$N_{\text{eXmY}}^{2^s\sigma} = 2 \times (s + \text{bias}(X) + 1) \times 2^Y + 1. \tag{18}$$

我们在表 1底部展示了某些浮点格式的 $N^{2^s}{}_{\text{eXmY}}$ 具体值。尽管低精度格式（如FP16）中的元素数量少于 FP32中的元素数量，但它仍大于现有的稀疏随机投影方法，后者通常为 3。

3.2.3 均值和方差

使用 RN 进行舍入时，g_{eXmY} 的均值为 0，因为浮点数值关于符号对称。另一方面，方差 $\alpha_Y = \mathbb{E}_g(g_{\text{eXmY}}{}^2) - (\mathbb{E}_g(g_{\text{eXmY}}))^2 = \mathbb{E}_g(g_{\text{eXmY}}{}^2)$，其中 $\mathbb{E}_x(y)$ 是 y 在 x下的均值，即使使用 $\mathcal{N}(0, 1)$ 生成数值并进行舍入，方差也不为 1。方差 α_Y 的计算如下：

$$\begin{aligned}\alpha_Y &= \mathbb{E}_g(g_{\text{eXmY}}{}^2) \\ &= \sum_{v \in \text{eXmY}} v^2 \left(\Phi(v + \text{exp_of}(v)u_Y) - \Phi(v - \text{exp_of}(v))u_Y\right)\end{aligned}$$

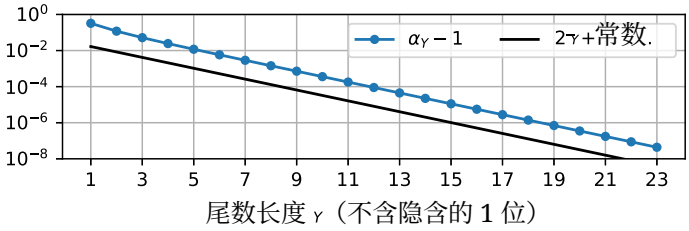


图 2：以 eXmY 表示的高斯分布的方差。

虽然无法对该方程进行解析计算，但我们进行了数值计算，如图 2 所示。当尾数长度为 Y 时，方差会指数级地趋近于 1。

虽然式 (4) 是使用 $\mathcal{N}(0<style id="2">, 1)$ 时的误差界，但我们证明，只要方差 α 不太小，该界就适用于任意方差。式 (4) 的证明可由下面这个公式的证明得出：

$$\mathbb{E}||\mathbf{\Sigma}_2\mathbf{G}_2\mathbf{G}_1^\dagger||_F^2 = k/(p-1)\cdot||\mathbf{\Sigma}_2||_F^2, \tag{19}$$

其中，Σ_2 是 $\mathrm{diag}(\sigma_{p+1},\cdots,\sigma_k)$，$\mathbf{G}_1, \mathbf{G}_2$ 分别是 $k\times k$和 $k\times(k-p)$ 高斯随机矩阵，$\cdot^\dagger$ 是一个矩阵的伪逆。假设高斯随机矩阵满秩，则可根据 [20]中的以下两个定理证明该公式。

定理 2 对于任意矩阵 $\mathbf{S}\in\mathbb{C}^{n_1\times n_2}$、$\mathbf{T}\in\mathbb{C}^{n_3\times n_4}$ 以及一个 $\mathcal{N}(0,1)$-高斯矩阵 $\mathbf{G}\in\mathbb{C}^{n_2\times n_3}$，

$$\mathbb{E}||\mathbf{SGT}||_F^2 = ||\mathbf{S}||_F^2||\mathbf{T}||_F^2. \tag{20}$$

定理 3 对于一个 $\mathcal{N}(0,1)$-高斯随机矩阵 $\mathbf{G}\in\mathbb{C}^{k\times(k+p)}$，

$$\mathbb{E}||\mathbf{G}^\dagger||_F^2 = k/(p-1). \tag{21}$$

由这些定理，式（19）证明如下。

$$\begin{aligned}
\mathbb{E}_{\mathbf{G}_1,\mathbf{G}_2}||\mathbf{\Sigma}_2\mathbf{G}_2\mathbf{G}_1^\dagger||_F^2 &= \mathbb{E}_{\mathbf{G}_1}\left(\mathbb{E}_{\mathbf{G}_2}\left[||\mathbf{\Sigma}_2\mathbf{G}_2\mathbf{G}_1^\dagger||_F^2\middle|\mathbf{G_1}\right]\right) && (22)\\
&= \mathbb{E}_{\mathbf{G}_1}\left(||\mathbf{\Sigma}_2||_F^2||\mathbf{G}_1^\dagger||_F^2\right) && \because \text{Theorem 2} \quad (23)\\
&= ||\mathbf{\Sigma}_2||_F^2\cdot\mathbb{E}_{\mathbf{G}_1}||\mathbf{G}_1^\dagger||_F^2 && (24)\\
&= k/(p-1)\cdot||\mathbf{\Sigma}_2||_F^2, && \because \text{Theorem 3} \quad (25)
\end{aligned}$$

其中 $\mathbb{E}[X|Y]$ 是在给定 Y 的条件下 X 的条件期望。

□

要证明当使用 $\mathcal{N}(0,\ \alpha)$ 时，式 (4)也成立，只需证明对于任意方差 α，式 (19) 成立。定理 2 和定理 3 扩展如下。

定理 4（定理 2 的扩展）对于任意矩阵 $\mathbf{S}\in\mathbb{C}^{n_1\times n_2}$、$\mathbf{T}\in\mathbb{C}^{n_3\times n_4}$ 以及一个 $\mathcal{N}(0,\alpha)$-高斯矩阵 $\mathbf{G}_\alpha\in\mathbb{C}^{n_2\times n_3}$，

$$\mathbb{E}||\mathbf{SG}_\alpha\mathbf{T}||_F^2 = \alpha||\mathbf{S}||_F^2||\mathbf{T}||_F^2. \tag{26}$$

定理 5（定理 3 的扩展）对于一个 $\mathcal{N}(0,\alpha)$-高斯随机矩阵$\mathbf{G}_\alpha\in\mathbb{C}^{k\times(k+p)}$，

$$\mathbb{E}||{\mathbf{G}_\alpha}^\dagger||_F^2 = k/(p-1)\cdot\alpha^{-1}. \tag{27}$$

我们将这些定理应用于公式（22-25），得到与公式（19）相同的方程。

□

定理 4 = α = α $\mathbb{E}|\ _{\alpha(i,j)}|^2$ = $\alpha\mathbb{E}|\ _{(i,j)}|^2$ $_{\cdot(i,j)}$ i j $_{(i,j)}$ $_{\alpha(i,j)}$ 的证明。对于 $\mathbf{M}$:$\mathbf{SGT}$ 和 $\mathbf{M}$:$\mathbf{SGT}$，只需证明$\mathbf{MM}$，其中 是矩阵的（，）元素。元素 $\mathbf{M}$和$\mathbf{M}$计算如下：

$$\mathbf{M}_{(i,j)} = \sum_{l_a}\sum_{l_b}\mathbf{S}_{(i,l_a)}\mathbf{G}_{(l_a,l_b)}\mathbf{T}_{(l_b,j)}$$
$$\mathbf{M}_{\alpha(i,j)} = \sum_{l_a}\sum_{l_b}\mathbf{S}_{(i,l_a)}\mathbf{G}_{\alpha(l_a,l_b)}\mathbf{T}_{(l_b,j)}.$$

因此， $\mathbb{E}|\mathbf{M}_{(i,j)}|^2$ 和 $\mathbb{E}|\mathbf{M}_{\alpha(i,j)}|^2$ 按如下方式计算：

$$\begin{aligned}
&\mathbb{E}|\mathbf{M}_{(i,j)}|^2 \\
&= \sum_{l_a,l_{a'},l_b,l_{b'}} |\mathbf{S}_{(i,l_a)}\mathbf{S}_{(i,l_{a'})}\mathbf{T}_{(l_b,j)}\mathbf{T}_{(l_{b'},j)}| \times \mathbb{E}\left(\mathbf{G}_{(l_a,l_b)}\mathbf{G}_{(l_{a'},l_{b'})}\right) && (28)\\
&= \sum_{l_a}\sum_{l_b}|\mathbf{S}_{(i,l_a)}|^2\cdot|\mathbf{T}_{(l_b,j)}|^2, && (29)\\
&\mathbb{E}|\mathbf{M}_{\alpha(i,j)}|^2 \\
&= \sum_{l_a,l_{a'},l_b,l_{b'}} |\mathbf{S}_{(i,l_a)}\mathbf{S}_{\ (i,l_{a'})}\mathbf{T}_{\ (l_b,j)}\mathbf{T}_{\ (l_{b'},j)}| \times \mathbb{E}\left(\mathbf{G}_{\ \alpha(l_a,l_b)}\ \mathbf{G}^{\alpha(l_{a'},l_{b'})}\right) && (30)\\
&= \alpha\sum_{l_a}\sum_{l_b}|\mathbf{S}_{(i,l_a)}|^2\cdot|\mathbf{T}_{(l_b,j)}|^2 = \alpha\mathbb{E}|\mathbf{M}_{(i,j)}|^2. && (31)
\end{aligned}$$

我们分别使用以下事实来推导式 (29) 和 (31)，它们分别由方差的定义以及来自独立分布的两个值的乘积得出。

$$\mathbb{E}\left(\mathbf{G}_{(i_1,j_1)}\mathbf{G}_{(i_2,j_2)}\right) = \begin{cases} 1 & (i_1=i_2 \& j_1=j_2) \\ 0 & \text{otherwise} \end{cases} \quad (32)$$

$$\mathbb{E}\left(\mathbf{G}_{\alpha(i_1,j_1)}\mathbf{G}_{\alpha(i_2,j_2)}\right) = \begin{cases} \alpha & (i_1=i_2 \& j_1=j_2) \\ 0 & \text{otherwise} \end{cases} \quad (33)$$

□

定理 *5* 的证明。对于一个高斯随机矩阵$\mathbf{G}\ \in \mathbb{R}^{m\times n}$，$(\mathbf{G}\mathbf{G}^\top)^{-1}$ 服从逆 Wishart 分布 [1]。逆 Wishart 分布的均值计算如下：

$$\mathbb{E}((\mathbf{G}\mathbf{G}^\top)^{-1}) = 1/(n-m-1)\cdot\mathbf{S}_{\mathbf{G}}^{-1}, \quad (34)$$

其中 $\mathbf{S}_{\mathbf{G}}$ 是$\mathbf{G}$的协方差矩阵。由于 $||\mathbf{G}^\dagger||_F^2$ =tr$[(\mathbf{G}^\dagger)^\top\mathbf{G}^\dagger]$ =tr$[(\mathbf{G}\mathbf{G}^\top)^{-1}]$ ，

$$\mathbb{E}||\mathbf{G}^\dagger||_F^2 = 1/(n-m-1)\cdot\mathrm{tr}[\mathbf{S}_{\mathbf{G}}^{-1}]. \quad (35)$$

当$\mathbf{G}$ 是一个 $k\times(k-p)$矩阵$\mathbf{G}_r^{\mathrm{Y}}$时，$\mathbf{S}_{\mathbf{G}\mathbf{G}}^\top$ 的非对角元素为零，对角元素为 α。因此，我们得到以下方程：

$$\mathbb{E}||\mathbf{G}^\dagger||_F^2 = k/(p-1)\cdot\alpha^{-1}. \quad (36)$$

□

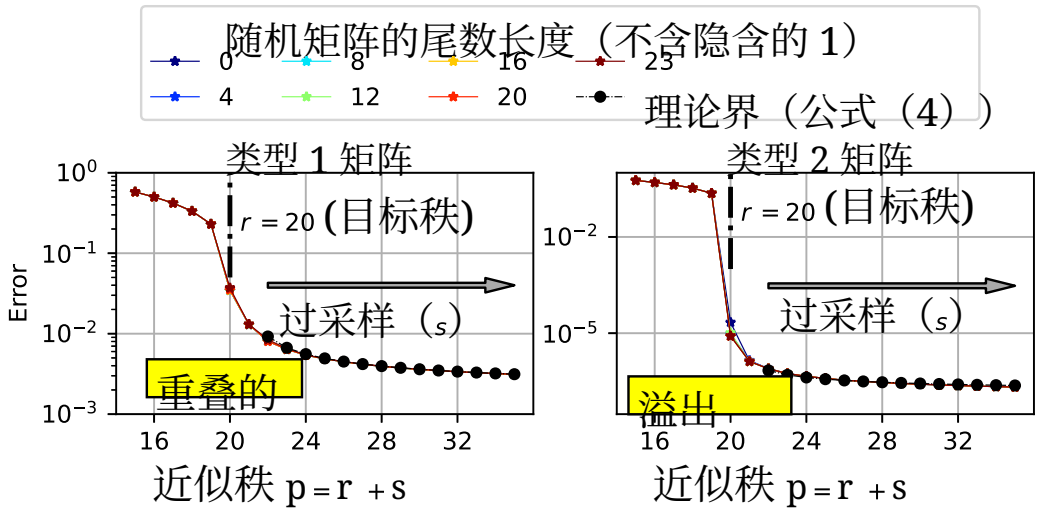


图 3：使用低精度高斯矩阵对两种输入矩阵进行随机投影的精度。误差计算为 $||\mathbf{A} - \mathbf{Q}\mathbf{Q}^\top\mathbf{A}||_F$。

3.3 数值实验

我们进行数值实验，以研究尾数长度对随机投影精度的影响。FP32 输入矩阵$\mathbf{A}_{t1}$ 和$\mathbf{A}_{t2}$ 按照以下两种方式生成，用于 [9] 中的数值实验。

- 类型 1：对于 $\mathbf{D} = \text{diag}(\mathbf{I}_r, 0) \in \mathbb{R}^{n\times n}$，我们定义

$$\mathbf{A}_{t1} = \mathbf{D} + (\xi/\epsilon)\mathbf{G}\mathbf{G}^\top,$$

  其中$\mathbf{G}$ 是一个高斯随机矩阵。我们固定 $\xi = 10^{-4}, n = 4096$ 和 $r = 20$。

- 类型 2：对于 Haar 分布中的正交随机矩阵 $\mathbf{U}$，$\mathbf{V} \in \mathbf{R}^{n\times n}$，一个 d

$$\mathbf{D} = \text{diag}(\phi\mathbf{I}, 2^{-\alpha}, 3^{-\alpha}, \cdots, (n-r+1)^{-\alpha}),$$

  我们定义 $\mathbf{A}_{t2} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$。我们固定 $n = 4096, r = 20, \alpha = 3$ 和 $\phi = 10^6$。

高斯随机矩阵 $\mathbf{G}$由FP32 生成，并通过RN舍入为低尾数长度值。为避免舍入误差的影响，我们将输入矩阵和随机矩阵都转换为FP64，并在FP64中计算随机投影。评估结果如图3所示。在两个测试矩阵中，随机投影的精度在所有尾数长度下几乎相同。

3.4 与现有稀疏随机投影方法的结合

现有的稀疏随机投影方法使用由式（5）生成的稀疏随机矩阵。使用稀疏随机矩阵$\mathbf{\Omega}$计算随机投影时，由于我们只使用投影矩阵的正交矩阵，因此无需乘以式（5）中的$\sqrt{n}$ 。因此，随机投影可以使用元素均为 $\{-1, 0, 1\}$ 之一的稀疏随机矩阵。即使尾数只有隐含的 1 位，也可以表示这些值。因此，混合精度随机投影方法也可以应用于（极）稀疏随机矩阵方法，并保持相同的精度。不过，本文其余部分仅关注高斯随机投影，因为从理论上讲，它是 RandNLA 中理解最透彻的一类随机矩阵 [20]。

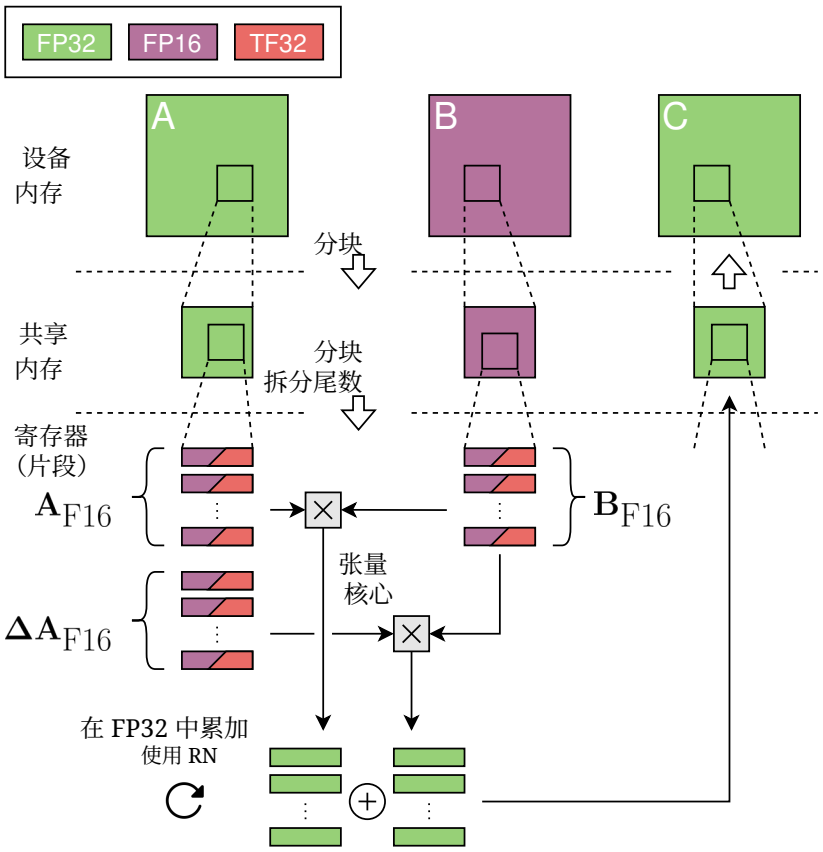


图 4：我们实现中 SHGEMM 在张量核心上的计算流程 .

4 张量核心上的单精度和半精度矩阵乘法

4.1 算法

通过在 TCEC-SGEMM 模拟方法中省略一项，我们可以按如下方式计算单精度矩阵$\mathbf{A}_{\mathrm{F32}}$ 与半精度矩阵$\mathbf{B}_{\mathrm{F16}}$ 的乘积，并达到单精度精度：

$$\mathbf{A}_{\mathrm{low}} \leftarrow \mathrm{toLow}\left(\mathbf{A}_{\mathrm{F32}}\right) \tag{37}$$

$$\Delta\mathbf{A}_{\mathrm{low}} \leftarrow \mathrm{toLow}\left(\left(\mathbf{A}_{\mathrm{F32}} - \mathrm{toF32}\left(\mathbf{A}_{\mathrm{low}}\right)\right) \times 2^{11}\right) \tag{38}$$

$$\mathbf{B}_{\mathrm{low}} \leftarrow \mathrm{toLow}\left(\mathbf{B}_{\mathrm{F16}}\right) \tag{39}$$

$$\mathbf{C}_{\mathrm{SH}} \sim \mathbf{A}_{\mathrm{low}} \cdot \mathbf{B}_{\mathrm{low}} + \Delta\mathbf{A}_{\mathrm{low}} \cdot \mathbf{B}_{\mathrm{low}} \times 2^{-11}. \tag{40}$$

由于该方法的计算复杂度是 $2/3$ TCEC-SGEMM 的计算复杂度，因此当矩阵 $\mathbf{B}_{\mathrm{F32}}$ 可以用 FP16 表示时，它可以更快地执行矩阵乘法。我们将该方法**SHGEMM** （单精度和半精度 GEMM）。

4.2 实现

我们使用 WMMA API 和 WMMA API 扩展库 [35] 从头实现了 SHGEMM 内核函数。使用 FP16 张量核心时，$\mathbf{A}_{\mathrm{F32}}$ 支持的指数范围仅限于 FP16 的指数范围。因此，我们实现了两种 SGEMM：1）有限指数范围 SHGEMM

使用 FP16 张量核心的 SHGEMM（SHGEMM-FP16），以及 2）使用 TF32 张量核心、支持完整指数范围的 SHGEMM（SHGEMM-TF32）。由于 FP16 张量核心的理论峰值性能比 TF32 张量核心 $2\times$ 高，因此 SHGEMM-FP16 理论上比 SHGEMM-TF32 更快。即使在 SHGEMM-TF32 的实现中，矩阵 $\mathbf{B}_{\mathrm{F16}}$ 也以 FP16 格式存储在设备内存中，在共享内存中以 FP16 格式加载和存储，并在输入张量核心之前于寄存器中转换为 TF32，如图 4 所示。该实现可在 GitHub 上获取[1]。

4.3 舍入误差分析

我们简要分析 SHGEMM 的舍入误差界，并表明它与在 FP32 SIMT 核心上进行计算时的误差界相同。我们使用 mma.m16n8k8 PTX 指令，该指令可在张量核心上执行矩阵乘法和加法。只需分析内积$c = \mathbf{x}^\top \mathbf{y} = \Sigma_{i=1}^{n} x_i y_i$的误差界即可。使用该 PTX 指令进行计算时的特性以及避免 RZ 的方法如下：

- 一条指令会累加乘法运算得到的八个结果值。
- 张量核心内部累加器的尾数长度为 25 位，其中包括隐藏位。
- 两个 11 位尾数值相乘时不会产生舍入误差，乘积会以 RZ 模式累加。
- 累加得到的八个值会经 RZ 舍入后以 FP32 格式输出。
- 八个值的结果累加在采用 RN 的 FP32 SIMT 核心上进行。

这些性质已通过理论分析或小型数值实验得到确认。内积被拆分为对八个元素进行累加的累积，具体如下：

$$c = \sum_{i=1}^{n'} t_i \quad \text{for} \quad t_i = \sum_{j=1+8(i-1)}^{8i} x_j y_j,$$

其中 $n' = n/8$。首先，我们得到两个 FP16向量的内积舍入误差界。˜我们分别将 $\tilde{c}$ 和 t_i 定义为在张量核心上计算 c 和 t_i 的结果，如下：

$$\tilde{c} = \mathrm{fl}_{RX}\left(\sum_{i=1}^{n'} \mathrm{RZ}_{F32}\left(\tilde{t}_i\right)\right), \quad \tilde{t}_i = \mathrm{fl}_{tc}\left(\sum_{j=1+8(i-1)}^{8i} x_j y_j\right),$$

其中，$\mathrm{fl}_{RX}(f)$ 表示使用RX对 f进行计算，其中舍入方式可以是RN 或 RZ。$\mathrm{fl}_{tc}(f)$在张量核心上执行，张量核心具有上述性质，而$\mathrm{RZ}_{F32}(\cdot)$ 是一个通过RZ 将25 位尾数舍入为FP32的舍入函数。在 $\mathrm{fl}_{RX}(f)$ 中，舍入误差的界如下：

$$\mathrm{RX}(a) = a(1+\delta), |\delta| \leq \hat{u} \equiv \begin{cases} u & \mathrm{RX\ is\ RN} \\ 2u & \mathrm{RX\ is\ RZ}, \end{cases} \tag{41}$$

其中，u 是 FP32 的单位舍入误差，2^{-24}。RZ 模态是张量核心上的默认模态，而 RN模态用于 SHGEMM，与TCEC-SGEMM 相同。此外，t_i 的舍入误差˜界定如下：

$$\tilde{t}_i = t_i + \Delta_i \ \text{ s.t. } \ |\Delta_i| \leq \underbrace{7}_{\text{Num additions}} \cdot \underbrace{2u'}_{\text{RZ}} \cdot \left(\sum_{j=1+8(i-1)}^{8i} |x_j y_j|\right),$$

[1] https://github.com/enp1s0/shgemm

其中 $u' = 2^{-25}$。于是，满足以下舍入误差方程。

$$\tilde{c} = \sum_{i=1}^{n'} \mathrm{RZ}_{F32}(\tilde{t}_i) + \Delta = \sum_{i=1}^{n'} (t_i + \Delta_i)(1 + \delta_i) + \Delta$$

其中， δ_i 是式 (41) 中 $\tilde{t}_i$ 对应的 δ，而 Δ 是满足 $|\Delta| \leq (n'-1)\hat{}u \sum_{i=1}^{n'} \mathrm{RZ}_{F32}(\tilde{t}_i)$ 的值。通过展开 δ_i、Δ_i 和 Δ 的不等式，我们得到如下内积误差界。

$$|\tilde{c} - c| = \left| \tilde{c} - \sum_{i=1}^{n'} t_i \right| \lesssim \left| \sum_{i=1}^{n'} (t_i \delta_i + \Delta_i(1 + \delta_1)) + \Delta \right| \lesssim \frac{1}{8} n\hat{u} |x^\top||y|,$$

其中， $|x^\top||y|$ 是 x 和 y 的逐元素绝对值向量的内积。接下来，我们将内积舍入误差的分析扩展到两个 FP16 矩阵在 FP32 中的乘法，$\mathbf{A}_{\mathrm{F16}} \cdot \mathbf{B}_{\mathrm{F16}}$，并得到如下误差界不等式：

$$|\tilde{\mathbf{C}} - \mathbf{A}_{\mathrm{F16}} \cdot \mathbf{B}_{\mathrm{F16}}| \lesssim \begin{cases} \frac{1}{8} nu |\mathbf{A}_{\mathrm{F16}}||\mathbf{B}_{\mathrm{F16}}| & \text{RX is RN} \\ \frac{1}{4} nu |\mathbf{A}_{\mathrm{F16}}||\mathbf{B}_{\mathrm{F16}}| & \text{RX is RZ,} \end{cases} \tag{42}$$

˜，对于 $n \geq 8$，其中 $\mathbf{C}$ 是计算结果， $|\cdot|$ 是逐元素绝对值 e.

接下来，我们利用张量核心的误差界以及上述对 RZ 的规避方法，推导 SHGEMM 的误差界。在式 (37-38) 中，矩阵 $\mathbf{A}_{\mathrm{F32}}$ 被拆分为以下三项：

$$\mathbf{A}_{\mathrm{F32}} = \mathbf{A}_{\mathrm{F16}} + \Delta\mathbf{A}_{\mathrm{F16}} + \mathbf{A}_{\Delta}, \tag{43}$$

其中，$\mathbf{A}_{\Delta}$ 表示拆分所造成的尾数损失，其期望长度为 0.25 位 [34]。对于拆分后的矩阵，满足以下不等式：

$$|\mathbf{A}_{\mathrm{F16}}| \leq (1 + u_{\mathrm{F16}})|\mathbf{A}_{\mathrm{F32}}|, |\Delta\mathbf{A}_{\mathrm{F16}}| \leq u_{\mathrm{F16}}|\mathbf{A}_{\mathrm{F32}}|, |\mathbf{A}_{\mathbf{\Delta}}| \leq u_{\mathrm{F16}}^2 |\mathbf{A}_{\mathrm{F32}}|, \tag{44}$$

其中， u_{F16} 表示 FP16 的单位舍入误差， 2^{-11}。我们按如下方式计算 SHGEMM：

$$\begin{aligned} \tilde{\mathbf{C}}_{\mathrm{SH}} &= \mathrm{fl}\left(\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}} + \Delta\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}}\right) && (45) \\ &= \mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}} + \mathbf{\Delta}^{(1)} + \Delta\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}} + \mathbf{\Delta}^{(2)} \\ &\quad + \delta(\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}} + \mathbf{\Delta}^{(1)} + \Delta\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}} + \mathbf{\Delta}^{(2)}), && (46) \end{aligned}$$

其中， $\mathbf{\Delta}^{(1)}$ 和 $\mathbf{\Delta}^{(2)}$ 分别是 $\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}}$ 和 $\Delta\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}}$中的舍入误差。我们使用 Oomoto 的 RZ 避免方法来计算$\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}}$，但不用于 $\Delta\mathbf{A}_{\mathrm{F16}}\mathbf{B}_{\mathrm{F16}}$。因此，根据公式 (42) 和 (44)，以下不等式成立：

$$|\mathbf{\Delta}^{(1)}| \lesssim \frac{1}{8} nu |\mathbf{A}_{\mathrm{F32}}||\mathbf{B}_{\mathrm{F16}}| \tag{47}$$

$$|\mathbf{\Delta}^{(2)}| \lesssim \frac{1}{4} nu u_{\mathrm{F16}} |\mathbf{A}_{\mathrm{F32}}||\mathbf{B}_{\mathrm{F16}}|. \tag{48}$$

因此，通过展开公式 (46)，我们得到 SHGEMM 舍入误差界的以下不等式：

$$|\tilde{\mathbf{C}}_{\mathrm{SH}} - \mathbf{A}_{\mathrm{F32}} \cdot \mathbf{B}_{\mathrm{F16}}| \lesssim \frac{1}{8} nu |\mathbf{A}_{\mathrm{F32}}||\mathbf{B}_{\mathrm{F16}}|, \tag{49}$$

对于 $n \geq 8$。在 FP32 SIMT 核心上以 8 个元素分块计算乘积 $\mathbf{A}_{\mathrm{F32}} \cdot \mathbf{B}_{\mathrm{F16}}$ 时，主要的误差界项同样为 $\frac{1}{8} nu |\mathbf{A}_{\mathrm{F32}}||\mathbf{B}_{\mathrm{F16}}|$。因此，SHGEMM 的误差界与在 FP32 SIMT 核心上的计算相同。

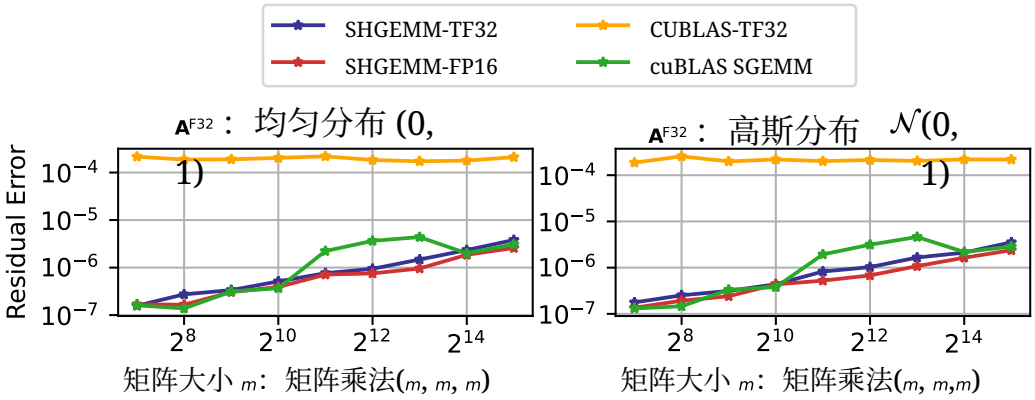


图 5：SHGEMM上矩阵乘法 $\mathbf{A}_{\mathrm{F32}} \cdot \mathbf{B}_{\mathrm{F16}}$ 的精度。矩阵$\mathbf{B}_{\mathrm{F16}}$ 由高斯分布 $\mathcal{N}(0,1)$ 生成，而$\mathbf{A}_{\mathrm{F32}}$ 服从 $\mathcal{N}(0,1)$（右）或均匀分布 (0,1)（左）。

4.4 SHGEMM 的评估

所有数值实验均在配备 NVIDIA A100 SXM4（40GB）图形处理器和 AMD EPYC 7702 中央处理器的系统上进行。

4.4.1 精度

SHGEMM 实现的精度如图 5 所示。我们计算乘积$\mathbf{A}_{\mathrm{F32}} \cdot \mathbf{B}_{\mathrm{F16}}$，其中$\mathbf{A}_{\mathrm{F32}}$使用 $\mathcal{N}(0,1)$ 或均匀分布 $(0,1)$ 进行初始化，而$\mathbf{B}_{\mathrm{F16}}$为$\mathcal{N}(0,1)$。此外，我们将矩阵$\mathbf{B}_{\mathrm{F16}}$ 转换为 FP32，以比较 cuBLAS SGEMM 和 cuBLAS TF32 GEMM 的精度。相对误差计算如下：

$$\mathrm{RelativeError} = ||\mathbf{C}_{\mathrm{SH}} - \mathbf{C}_{\mathrm{F64}}||_F / ||\mathbf{C}_{\mathrm{F64}}||_F,$$

其中$\mathbf{C}_{\mathrm{F64}}$ 是将输入矩阵转换为 FP64 后在 FP64 中计算得到的。SHGEMM 实现的精度与 cuBLAS SGEMM 处于同一水平。

4.4.2 吞吐量

SHGEMM 实现的吞吐量如图 6 所示。对于 matmul-(m,n,k)，吞吐量计算为$2mnk/t$ [Flop/s]，该操作计算 $m \times k$ 和 $k \times n$ 矩阵的乘积，其中 t 是计算时间，单位为 [sec]。我们比较了两种 SHGEMM 实现、使用 FP16 张量核心和 TF32 张量核心的 TCEC-SGEMM 实现，以及 cuBLAS SGEMM 的吞吐量。SHGEMM-FP16 的吞吐量最高可达 61.0 [每秒万亿次浮点运算]，，SHGEMM-TF32 的吞吐量为 40.5 [每秒万亿次浮点运算]。每种 SHGEMM 实现的速度都快于使用相同张量核心的 TCEC-SGEMM。

在以下假设下，NVIDIA A100 图形处理器上 SHGEMM-FP16 和 SHGEMM-TF32 的理论峰值性能分别为 $312/2 = 156$ [每秒万亿次浮点运算] 和 $152/2 = 76$ [每秒万亿次浮点运算]，。

假设1除张量核心之外的计算单元上的所有计算都可以与张量核心上的计算重叠。换句话说，张量核心指令每个时钟周期都会发出。

因此，SHGEMM-FP16 的效率仅为 39%，而 SHGEMM-TF32 的效率为 53%。我们认为，效率低下的原因在于难以实现假设 1，具体原因如下。

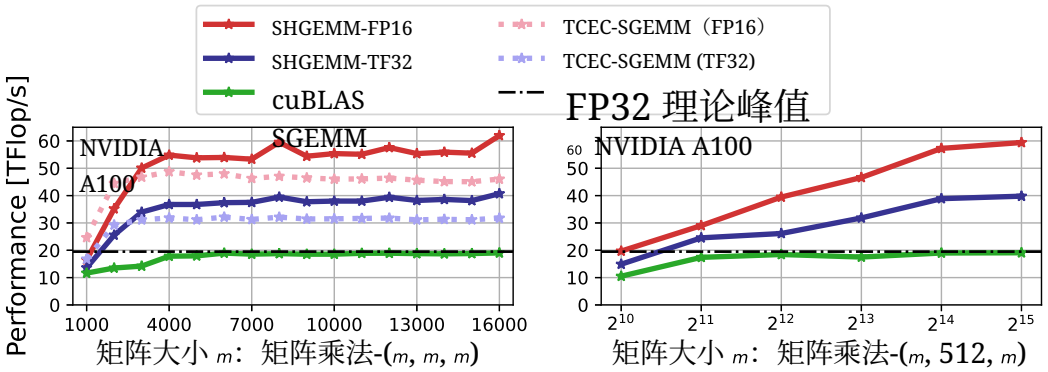


图 6：左：cuBLAS SGEMM、TCEC-SGEMM 和 SHGEMM 的吞吐量比较。后缀 FP16 和 TF32 表示 GEMM 实现中使用的 Tensor Core 类型。右：用于 $m \times m$ 矩阵的秩 -512 随机奇异值分解中高瘦矩阵乘法的 SHGEMM 吞吐量比较。

1. 式 (37-40) 中，张量核心和 FP32 SIMT 核心的计算顺序存在依赖关系。因此，这些计算无法重叠。

2. 在 FP32 SIMT核心上计算公式(38)以及避免TensorCore 内部的 RZ 所需的计算时间很长。当将矩阵乘法(m, n, k)得到的矩阵划分为网格，使每个矩阵的矩阵大小为 $b_m \times b_n$，并行计算每个细分矩阵的结果时，加载的 $\mathbf{A}_{\mathrm{F32}}$ 元素数量为 mnk/b_n。因此，公式 (38)中减法和乘法的数量为 $2mnk/b_n$，并在 FP32SIMT核心上计算。此外，我们使用 FP32 SIMT核心累加细分矩阵乘法的结果，以避免 Tensor Core 内部的RZ $2mnk/f_k$ 次，其中 f_k 是一次张量核心指令所计算的矩阵乘法的 k 值，该值为8。另一方面，公式(40) 中在 Tensor Core 上的计算数量为 $4mnk$。因此，FP32SIMT 核心和 Tensor Core 上的计算比率为 $(2/b_n + 1/f_k) : 4$，与 m，n 和 k无关。由于我们通常将 $b_n = 64$设为 128，因此该比率约为 $1 : 25$。因此，相对于 Tensor Core 上的计算时间，FP32SIMT 核心上的计算时间不可忽略，因为在 FP16 张量核心上，FP32SIMT 核心与 TensorCore的理论吞吐量比仅为 $1 : 16$，而在 TF32 张量核心上仅为 $1 : 8$。

5 实验

5.1 随机化奇异值分解

我们在算法 1 的第 1 行使用 SHGEMM，并在算法其余部分的计算中使用 NVIDIA cuBLAS 和 cuSOLVER 库。

5.1.1 精度

为评估精度，我们使用以下 4 种测试矩阵：

- $\mathbf{A}_{\text{线性}}$ 和 $\mathbf{A}_{\text{指数}} \in \mathrm{FP32}^{4096\times 4096}$：首先，我们确定一个值 $s_p \in (0,1)$，并生成如下 FP32 值序列 $\{s_j\}$。－对于 $\mathbf{A}_{\text{线性}}$：$s_i = \max(-\alpha_l \times i + 1, s_p)$，其中 $\alpha_l = (1 - s_p)/p$。－对于 $\mathbf{A}_{\text{指数}}$：$s_i = 2^{-\alpha_e \times i}$，其中 $\alpha_e = (\log_2 1/s_p)/p$。

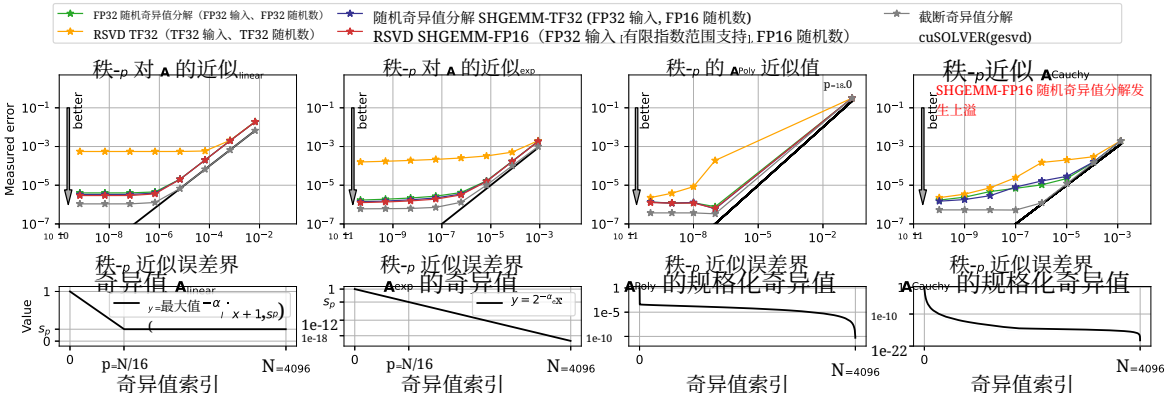


图 7：**上：** 针对输入矩阵 $\mathbf{A}_{\text{linear}}$、$\mathbf{A}^{\text{exp}}$和 $\mathbf{A}^{\text{Cauchy}}$的随机化奇异值分解实现的精度。 **下：** 输入矩阵的奇异值衰减和目标秩。

然后，我们使用 LAPACK slatms 函数生成一个 $N \times N$ FP32 随机矩阵，其奇异值为 s_i。通过这种方式生成矩阵，我们可以使用定理 1 计算 p-秩近似的误差界。因此， p-秩近似的相对残差误差界对于 $\mathbf{A}_{\text{linear}}$ 为

$$s_p\sqrt{N-p}/||\mathbf{A}_{\text{linear}}||_F$$

以及 $\mathbf{A}^{\text{exp}}$ 是

$$\sqrt{(s_p^2 - 2^{2qN})/(1-2^{2q})}/||\mathbf{A}_{\text{exp}}||_F.$$

在此评估中， N 和 p 固定为 4096 和 256。固定这些参数后，随机化奇异值分解的计算复杂度也随之固定。因此，舍入误差累积在一定程度上也固定下来，我们可以比较随机投影精度对近似精度的影响。

- $\mathbf{A}_{\text{多项式}} \in \text{FP32}^{4096\times4096}$：生成公式与 3.3 中的矩阵类型 2 相同。我们使用了参数 $r = 20, \alpha = 3$ 和 $\phi = 10^6$。
- $\mathbf{A}_{\text{柯西}} \in \text{FP32}^{4096\times4096}$：我们按如下方式生成柯西矩阵，并进行 1 次正交迭代。

$$\mathbf{A}_{\text{Cauchy}_{i,j}} = 1/\left(|x_i - y_i| + \gamma\right),$$

其中 x_i, y_i 是 $(-10^{-3}, 10^{-3})$ 中的随机序列，且 $\gamma = 10^{-3}$。该矩阵的一些元素的绝对值超出了 FP16 的表示范围。因此，使用 SHGEMM-FP16 时，计算预计会失败。

我们固定过采样参数 $s = 10$ ，并评估使用不同随机种子生成的十个矩阵。我们比较了不同 GEMM 实现下随机化奇异值分解的精度，如图 5 所示。使用 SHGEMM 时，随机矩阵的数据类型为 FP16；否则为 FP32。与作为基准的 FP32 随机奇异值分解相比，RSVD TF32 的精度有所下降，因为 TF32 GEMM 的精度低于 SGEMM。这表明，在随机投影过程中保持输入矩阵的精度至关重要。另一方面，RSVD SHGEMM-TF32 和 SHGEMM-FP16 的精度与 FP32 随机奇异值分解处于同一水平。

5.1.2 吞吐量

随机化奇异值分解实现的吞吐量比较如图 8 所示。尽管随机投影的渐近计算复杂度高于奇异值分解和 QR 分解，但它们的实际计算时间也不可忽略。因此，在 $m = n = 8192$， $rank = 512$ 的情况下，

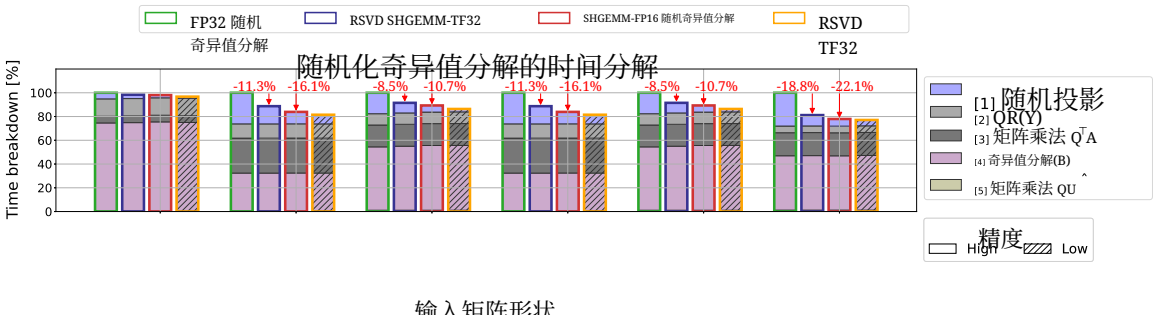


图 8：NVIDIA A100 上基线随机化奇异值分解（FP32）的时间分解，以及我们的方法带来的计算时间缩减。分解标签前的数字是算法 1 中的行号。

算法2随机投影高阶奇异值分解

要求：输入矩阵A $\in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$，**目标秩** $r \in \mathbb{N}$ **确保：** $\mathbf{g} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$，$\mathbf{Q}_k \in \mathbb{R}^{I_k \times J_k}$ **使得** $\mathbf{A} \sim \mathbf{g} \times_1 \mathbf{Q}_1 \times_2 \mathbf{Q}_2 \cdots \times_N \mathbf{Q}_N$**1：对于 do** $i = 1..N$**2：W** $\leftarrow \mathbf{A}'_{(i)} \cdot \mathbf{\Omega}^{(i)}$ **其中** $\mathbf{A}'_{(i)} \in \mathbb{R}^{I_i \times (\Pi_{k \neq i} I_k)}$**是张量A 的转置矩阵，且**$\mathbf{\Omega}_{(i)} \in \mathbb{R}^{(\Pi_{k \neq i} I_k) \times J_i}$ **是随机矩阵。3：**$\mathbf{Q}_i$，$\leftarrow$ **QR分解 (W)4：结束循环 5：**$\mathbf{g} \leftarrow \mathbf{A} \times_1 \mathbf{Q}_1^\top \times_2 \mathbf{Q}_2^\top \cdots \times_N \mathbf{Q}_N^\top$

由于它们的计算时间约占总计算时间的 90%，我们无法加速整个计算过程。另一方面，当奇异值分解和 QR 分解的计算时间占比不太大时，使用 SHGEMM-TF32 和 SHGEMM-FP16 分别可将总计算时间最多缩短 18.9% 和 22.1%，同时精度与 FP32 处于同一水平。

5.2 随机投影高阶奇异值分解

我们评估了我们的方法对高阶奇异值分解（HOSVD）的影响 [11]。HOSVD 将一个 N 维张量 $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ 分解为一个 N 维张量与矩阵的乘积，如下所示：

$$\mathbf{A} = \mathbf{g} \times_1 \mathbf{Q}_1^\top \times_2 \mathbf{Q}_2^\top \cdots \times_N \mathbf{Q}_N^\top, \tag{50}$$

其中，$\mathbf{g} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ 是一个核心张量，$\mathbf{Q}_k \in \mathbb{R}^{I_k \times J_k}$ 是一个正交矩阵。我们将张量 $\mathbf{T} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ 与矩阵 $\mathbf{M} \in \mathbb{R}^{I_i \times k}$ 在第 i 个模态上的收缩记为 $\mathbf{T} \times_i \mathbf{M}$。每个维度上的秩决定了核心张量的形状。HOSVD 通过将张量展平为矩阵并进行奇异值分解来计算。算法2所示的随机投影高阶奇异值分解（RP-HOSVD） [3]使用随机投影和 QR 分解代替奇异值分解来计算该分解。

为评估 RP-HOSVD，我们按照算法 3 生成测试张量，并测量近似精度和吞吐量，如图 9 所示。与随机化奇异值分解的精度情况一样，使用 TF32随机投影时，精度会下降。另一方面，使用 SHGEMM-TF32 和 SHGEMM-FP16 进行随机投影时，计算时间分别缩短了 36.6% 和 43.0%。尽管在某些情况下，采用 SHGEMM 的 RP-HOSVD 精度优于 FP32 基线，但我们认为这是计算顺序不同所致，这一点也体现在图 5 中 SHGEMM 的单元测试结果上。因此，我们得出结论：采用 SHGEMM 的 RP-HOSVD 精度与 FP32 基本处于同一水平。

算法 3用于评估 RP-HOSVD 的张量生成

需要： 输入张量维度 $I_i \in \mathbb{N}$，目标秩 $J_i \in \mathbb{N}$，秩填充 $p \in \mathbb{N}$

确保： 输入张量 $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$

$\mathbf{G} \leftarrow \text{GenRand}(-1,1) \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$

for do $i = 1..N$

$\mathbf{\Omega}_\alpha \leftarrow \text{GenRand}(-1,1) \in \mathbb{R}^{J_i \times (J_i - p)}$

$\mathbf{\Omega}_\beta \leftarrow$ 生成随机数$(-1,\ 1) \in \mathbb{R}^{(J_i - p) \times I_i}$

$\mathbf{\Omega}^{(i)} \leftarrow \mathbf{\Omega}_\alpha \cdot \mathbf{\Omega}_\beta$ // 这是一个（$J_i - p$）-秩矩阵

$\mathbf{G} \leftarrow \mathbf{G} \times_i \mathbf{\Omega}^{(i)}$

结束 for 循环

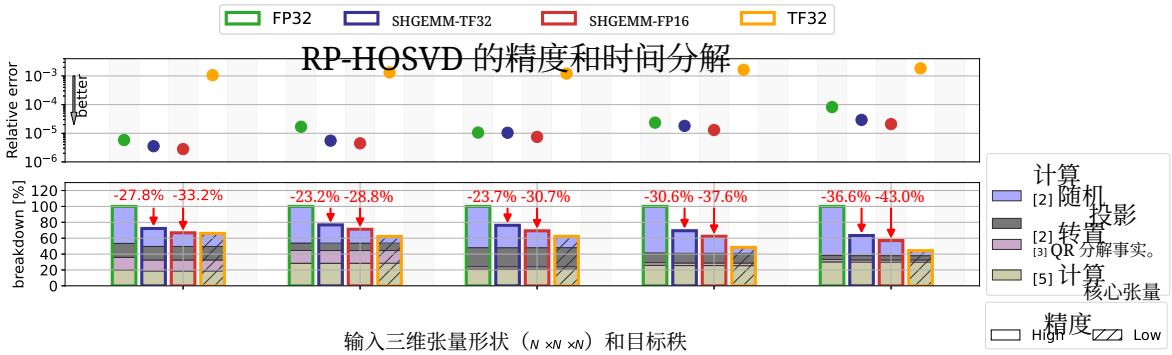


图9：NVIDIA A100 上随机投影高阶奇异值分解的精度（上）和时间分解（下）。分解标签前的数字是算法2中的行号。

6 结论

我们提出了一种适用于单精度张量的快速混合精度随机投影方法，在保持近似精度的同时利用张量核心的高吞吐量。我们验证了低精度高斯随机值适用于随机投影。用于随机投影的矩阵乘法 SHGEMM 使用 FP16 张量核心时可达到 61.0 每秒万亿次浮点运算，使用 TF32 张量核心时可达到 40.5 每秒万亿次浮点运算。与基线 FP32 实现相比，使用该方法可将随机化奇异值分解的吞吐量提高 1.28 倍，将随机投影高阶奇异值分解的吞吐量提高 1.75 倍，同时保持精度。

致谢

作者谨向 Katsuhisa Ozaki 教授致谢，感谢其提供技术协助并参与讨论。本研究获得 JSPS KAKENHI 项目资助，项目编号为 JP20K20624、JP22H03598 和 JP21J14694。本研究还获得日本跨学科大型信息基础设施联合使用／研究中心的支持（项目编号：jh220022-NAH、jh220009-NAHI）。

参考文献

[1] 多元统计理论专题。威利概率与统计丛书，霍博肯（美国新泽西州），1982年3月。John Wiley & Sons, Inc.

[2] Dimitris Achlioptas。数据库友好型随机投影：使用二元随机数的 Johnson-Lindenstrauss。《计算机与系统科学杂志》，66(4):671–687，2003 年。

[3]SalmanAhmadi-Asl、Stanislav Abukhovich、Maame G. Asante-Mensah、Andrzej Cichocki、Anh Huy Phan、Tohishisa Tanaka和IvanOseledets。用于计算 Tucker 分解和高阶奇异值分解（高阶奇异值分解）的随机化《算法》。*IEEE Access*，9：28684–28706，2021。会议名称：IEEE Access。[4] AMD。AMD CDNATM 2 架构，2021。[5] AshimaAnand 和 Amit KumarSingh。使用带模糊推理系统的IWT-Schur-随机奇异值分解进行基于云的安全水印，以应用于智慧医疗。可持续城市与社会，75：103398，2021 年 12 月。[6] MaolinChe、Yimin Wei和 Hong Yan。用于张量低多线性秩近似的随机化《算法》。计算与应用数学杂志，390：113380，2021。[7] Chuangquan Chen、Chi-Man Vong、Chi-Man Wong、Weiru Wang和 Pak-Kin Wong。通过非常稀疏的随机投影实现高效的极限学习机。软计算，22（11）：3563–3574，2018 年 6 月。[8] Chung-Cheng Chiu、James Qin、Yu Zhang、JiahuiYu 和 Yonghui Wu。使用随机投影量化器进行语音识别的自监督学习。载于第 *39*届国际机器学习会议论文集，第3915–3924页。PMLR，2022 年 6月。ISSN：2640-3498。[9] Michael P. Connolly、NicholasJ.Higham 和 Srikara Pranesh。随机低秩矩阵近似：舍入误差分析与混合精度《算法》，2022 年7 月。期号：2022.10 编号：2022.10。[10] NVIDIA公司。NVIDIA A100 张量核心图形处理器，2020。[11] Lieven De Lathauwer、BartDeMoor 和 Joos Vandewalle。多线性奇异值分-解。*SIAM* 矩阵分析与应用杂志，21（4）：1253–1278，2000 年 1 月。出版商：工业与应用数学学会。[12] Petros Drineas和Michael W. Mahoney。RandNLA：随机化数值线性代数。*ACM* 通讯，59（6）：80–90，2016。[13] Carl Eckart 和 Gale Young。用另一个低秩矩阵近似一个矩阵。心理测量学，1（3）：211–218，1936年 9 月。[14] N. Benjamin Erichson、StevenL. Brunton和J.Nathan Kutz。用于图像和视频处理的压缩奇异值分解。载于*2017* 年*IEEE* 国际计算机视觉会议研讨会（*ICCVW*），第1880–1888 页，2017 年 10 月。ISSN：2473-9944。[15] N. Benjamin Erichson、Lionel Mathelin、J.NathanKutz 和 Steven L. Brunton。随机化动态模态分解。*SIAM*应用动力系统杂志，18（4）：1867–1891，2019。出版商：工业与应用数学学会。[16] Xu Feng、WenjianYu和 Yaohang Li。使用随机化奇异值分解加速矩阵补全。载于*2018* 年 *IEEE*第*30* 届国际人工智能工具会议（*ICTAI*），第 608–615 页，2018 年 1 月。ISSN：2375-0197。[17] XiaoliZhang Fern 和 CarlaE. Brodley。高维数据聚类的随机投影：一种聚类集成方法。载于第二十届国际会议暨国际机器学习会议论文集，ICML’03，第 186–193 页，美国华盛顿特区，2003 年。AAAI Press。

[18] Dmitriy Fradkin 和 David Madigan。用于机器学习的随机投影实验。载于第九届 *ACM SIGKDD* 知识发现与数据挖掘国际会议论文集，KDD '03，第 517–522 页，美国纽约州纽约，2003 年。计算机协会。[19] Ming Gu 和 Stanley C. Eisenstat。高效计算强秩揭示QR分解的算法。《*SIAM*科学计算期刊》，2012 年 2 月。出版方：工业与应用数学学会。[20] N. Halko、P. G. Martinsson 和 J. A. Tropp。以随机性发现结构：用于构造近似矩阵分解的概率算法。《*SIAM*评论》，53(2)：217–288，2011 年。出版方：工业与应用数学学会。[21] Chinmay Hegde、Michael Wakin 和 Richard Baraniuk。用于流形学习的随机投影。载于神经信息处理系统进展，第 20 卷。Curran Associates, Inc.，2007 年。[22] Siriwan Intawichai 和 Saifon Chaturantabut。使用随机化奇异值分解与加速最小二乘近似的缺失数据重建方法。《算法》，15(6)：190，2022 年 6 月。期号：6。出版方：多学科数字出版研究所。[23] Norman P. Jouppi、Cliff Young、Nishant Patil 等。数据中心中张量处理单元的性能分析。载于第 *44* 届年度国际计算机体系结构研讨会论文集，ISCA '17，第 1–12 页，美国纽约州纽约，2017 年 6 月。计算机协会。[24] Saehoon Kim 和 Seungjin Choi。用于局部敏感二进制编码的双线性随机投影。第 1338–1346 页，2015 年。[25] Mu Li、Wei Bi、James T. Kwok 和 Bao-Liang Lu。使用随机化奇异值分解进行大规模 Nystr¨om 核矩阵近似。《*IEEE*神经网络与学习系统汇刊》，26(1)：152–164，2015 年 1 月。会议名称：《IEEE神经网络与学习系统汇刊》。[26] Ping Li、Trevor J. Hastie 和 Kenneth W. Church。极稀疏随机投影。载于第 *12* 届 *ACM SIGKDD* 知识发现与数据挖掘国际会议论文集，KDD '06，第 287–296 页，美国纽约州纽约，2006 年。计算机协会。[27] Junichiro Makino。现在、过去与未来。载于 Junichiro Makino（编），《高性能处理器设计原理：面向高性能计算、深度神经网络和数据科学》，第 147–155 页。施普林格国际出版，Cham，2021 年。[28] Stefano Markidis、Steven Wei Der Chien、Erwin Laure、Ivy Bo Peng 和 Jeffrey S. Vetter。NVIDIA Tensor Core 的可编程性、性能与精度。*2018 IEEE* 国际并行与分布式处理研讨会（*IPDPSW*），第 522–531 页，2018 年 5 月。arXiv：1803.04014。[29] Brendan McCane。用于特征匹配的随机投影。载于第 *29* 届新西兰图像与视觉计算国际会议论文集，IVCNZ '14，第 78–83 页，美国纽约州纽约，2014 年。计算机协会。[30] Vineetha Menon、Qian Du 和 James E. Fowler。用于高光谱降维的快速奇异值分解与随机哈达玛投影。《*IEEE*地球科学与遥感快报》，13(9)：1275–1279，2016 年 9 月。会议名称：《IEEE地球科学与遥感快报》。

[31] Rachel Minster、Arvind K. Saibaba 和 Misha E. Kilmer。Tucker 格式低秩张量分解的随机化算法。《数据科学数学期刊》，2(1)：189–215，2020 年。出版方：工业与应用数学学会。[32] Daichi Mukunoki、Katsuhisa Ozaki、Takeshi Ogita 和 Toshiyuki Imamura。使用张量核心执行 DGEMM 及其准确且可复现的版本。载于 Ponnuswamy Sadayappan、Bradford L. Chamberlain、Guido Juckeland 和 Hatem Ltaief（编），高性能计算，计算机科学讲义，第 230–248 页，Cham，2020 年。施普林格国际出版。[33]Mahmoud Nabil。随机投影及其应用，2017 年 10 月。编号：arXiv:1710.03163 arXiv:1710.03163 [cs]。[34] Hiroyuki Ootomo 和 Rio Yokota。借助张量核心恢复单精度精度，同时超越 FP32 理论峰值性能：《国际高性能计算应用期刊》，2022 年 6 月。出版方：SAGE PublicationsSage UK：伦敦，英格兰。[35] Hiroyuki Ootomo 和 Rio Yokota。通过减少共享内存占用，充分利用张量核心的高吞吐量及其灵活的 API 扩展库。载于亚太地区高性能计算国际会议论文集，HPC Asia '23，第 1–8 页，美国纽约州纽约，2023 年 2 月。计算机协会。[36] Katsuhisa Ozaki、Takeshi Ogita、Shin'ichi Oishi 和 Siegfried M. Rump。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。《数值算法》，59(1)：95–118，2012 年 1 月。[37]Fabian Pedregosa、Ga¨el Varoquaux、Alexandre Gramfort、Vincent Michel、Bertrand Thirion、Olivier Grisel、Mathieu Blondel、Peter Prettenhofer、Ron Weiss、Vincent Dubourg、Jake Van-´derplas、Alexandre Passos、David Cournapeau、Matthieu Brucher、Matthieu Perrot 和 Edouard Duchesnay。Scikit-learn：Python 机器学习。《机器学习研究期刊》，12(85)：2825–2830，2011 年。[38] Vladimir Rokhlin 和 Mark Tygert。用于超定线性最小二乘回归的快速随机化算法。《美国国家科学院院刊》，105(36)：13212–13217，2008 年 9 月。出版方：美国国家科学院院刊。[39] Gil Shabat、Yaniv Shmueli、Yariv Aizenbud 和 Amir Averbuch。随机化 LU 分解。《应用与计算谐波分析》，44(2)：246–272，2018 年 3 月。[40] O. P. Singh、Amit Kumar Singh、Amrit Kumar Agrawal 和 Huiyu Zhou。SecDH：基于 PCA 数据隐藏的 COVID-19 图像安全保护。《计算机通信》，191：368–377，2022 年 7 月。[41] Grigorios Tsagkatakis 和 Andreas Savakis。资源受限条件下用于人脸检测的随机投影。载于*2009* 年第 *16* 届 *IEEE* 图像处理国际会议（*ICIP*），第 1233–1236 页，2009 年 1 月。ISSN：2381-8549。[42] Nguyen Xuan Vinh、Sarah Erfani、Sakrapee Paisitkriangkrai、James Bailey、Christopher Leckie 和 Kotagiri Ramamohanarao。使用随机投影训练鲁棒模型。载于*2016* 年第 *23* 届国际模式识别会议（*ICPR*），第 531–536 页，2016 年 2 月。[43] Han Wang 和 James Anderson。使用随机化奇异值分解进行大规模系统辨识。载于*2022* 年美国控制会议（*ACC*），第 2178–2185 页，2022 年 6 月。ISSN：2378-5861。[44]Mingrui Yang、Dan Ma、Yun Jiang、Jesse Hamilton、Nicole Seiberlich、Mark A. Gris-wold 和 Debra McGivney。采用大规模字典的 MR 指纹识别低秩近似方法。《医学磁共振》，79(4)：2392–2400，2018 年。电子版：https://onlinelibrary.wiley.com/doi/pdf/10.1002/mrm.26867。

[45] Jiani Zhang、JenniferErway、Xiaofei Hu、Qiang Zhang 和RobertPlemmons。高光谱成像中的随机奇异值分解方法。*Journal of Electrical and Computer Engineering*，2012:3:3，2012。
[46] Jingyi Zhang、Ping Ma、Wenxuan Zhong 和 Cheng Meng。面向高维最优传输问题的基于投影的技术。*WIREs Computational Statistics*，页码 e1587。