arXiv:2603.20889v1 [cs.MS] 21 Mar 2026

当前 GPU 上高而极窄矩阵的 QR 分解实现

Jonas Thies[1][0000-0001-9231-9999] 和
Melven Röhrig-Zöllner[2][0000-0001-9851-5886]

[1] 代尔夫特理工大学 代尔夫特应用数学研究所 梅克尔韦赫4号，2628 CD 代尔夫特，荷兰 J.Thies@TUDelft.nl[2] 德国航空航天中心 软件技术研究所 林德高地，51147 科隆，德国 Melven.Roehrig.Zoellner@DLR.de

摘要。 我们研究实数、稠密、高而极窄矩阵的 QR（或 QZ）分解计算问题。也就是说，列数相较于行数非常少，使得大多数计算完全或部分受内存带宽限制。本文聚焦于仍支持 64位浮点运算的最新 NVIDIA 通用 GPU，但所得结论同样适用于 AMD GPU。我们讨论两种基本算法：基于正规方程（格拉姆矩阵）的方法，尤其是 Cholesky-QR2 和 SVQB；以及基于Householder变换、采用树形归约方案的“高瘦QR”（TSQR）。我们提出两种主要优化技术：避免回写 Q因子（“无 Q 的 QR”），以及利用快速本地内存（GPU 上的共享内存）。我们比较了直接实现的基于格拉姆矩阵的方法与更复杂的 TSQR 实现，考察其性能、求解时间和实现复杂度。通过性能建模，并结合使用自有代码和供应商优化的库例程进行的数值实验，我们证明，在这一内存受限至过渡型（内存/计算受限）区域中，专用方法与实现至关重要；TSQR 在求解时间方面具有竞争力，但需要投入精力进行底层代码优化。

关键词： TSQR · Cholesky-QR2 · 通信规避算法 · 图形处理器实现 · 无 Q 的 QR · 性能工程 · 屋顶线模型 · 内存受限极限

1 引言

我们考虑计算实稠密矩阵 $X \in \mathbb{R}^{m\times n}$ 的 QR分解问题，其中 $m \gg n$。这种矩阵通常被称为“高而窄”。高而窄 QR 有许多应用场景，例如计算

迭代线性求解器或特征值求解器中子空间的正交基，作为张量分解的构建模块，以及用于数据拟合或压缩。

形式上，我们希望计算 $Q \in \mathbb{R}^{m \times n}$ 和 $R \in \mathbb{R}^{n \times n}$，使得 $X = QR$、$Q^T Q = I$ 和 R 为上三角矩阵。如果 X 秩亏（$r = \text{rank}(X) < n$），R 将有一些对角线元素为零。针对高而窄情形的专用算法包括基于正规方程（Gram 矩阵）的方法：SVQB 的变体 [30] 和 Cholesky-QR[16]，以及基于 Householder变换的“高而窄 QR”（TSQR [7]）。SVQB 产生分解 $XB = Q$，$Q^T Q = I$，该分解可转换为 QZ 分解：$X = QZ$，$Z = \Sigma U$。其中，Σ，Q，而 U分别是 X的奇异值以及右（左）奇异向量。

这类算法统称为“通信规避”方法，因为它们旨在减少内存与计算单元之间，以及集群节点之间的数据流量。对于列数相对较少的矩阵，这可以缩短求解时间，因为所需的浮点计算无法摊销数据移动的开销。

本文特别关注高矩阵与**极**窄矩阵的情形，此时列数很少，内存流量主导计算时间。我们关注无 Q 的 QR算法变体，也就是说，这些算法完全不返回 Q 或其生成系数。如果 X 条件良好，则可在之后需要时重新构造 Q 。该术语由 Fan 等人提出 [12]，例如适用于 TT-SVD [27]等大型张量分解算法。它还可用于求解最小二乘问题 $\min_x \|Ax - b\|_2$：只需对扩展矩阵 $\begin{pmatrix} A & b \end{pmatrix}$进行一次遍历，并计算其无 Q 的 QR分解。我们的实现专注于在当前 NVIDIA GPU 上实现高性能，主要优化技术是在共享内存中进行手动缓存。相关结果也应适用于其他类型的 GPU（例如 AMD），前提是它们支持双精度运算，且每个多处理器都提供快速本地内存。

2 相关工作

各种方法在精度和稳定性方面的数值结果可参见经典教材 [17, 19]。标准豪斯霍尔德 QR 的运算可合并为三级 BLAS 运算，从而得到快速的 LAPACK 例程 dgeqrf。但只有采用适当的主元选取，才能确保无条件后向稳定性 [6]。遗憾的是，主元选取会显著降低性能，因为它通常需要受带宽限制的二级 BLAS 运算（LAPACK 例程 dgeqp3）。随着内存带宽与浮点性能之间的差距不断扩大（即冯·诺依曼瓶颈），这种性能损失在新一代 CPU 和 GPU 上愈发明显。因此，研究人员尝试克服主元选取的开销，例如通过无主元 QR 求解多个移位线性方程组 [4]，或在 QR 计算中跳过较小的列 [29]。对于某些性质良好的特定问题，这些方法似乎颇有前景，但一般而言可能不具备稳定性。具体来说，即使采用主元选取，将 QR分解的三角因子中的小对角元素“截断”所导致的最大

误差也会随秩 r 的 $O(2^r)$ 增长，参见 [18, 20]。计算奇异值分解时，通常会先对矩形高瘦矩阵执行无主元 Householder-QR正交化步骤，参见 [32]。

高瘦矩阵是一个重要情况，在这种情况下，数据传输比其他任何因素都更能决定运行时间，这推动了所谓通信规避算法的发展。TSQR（高瘦QR）和 CAQR（通信规避QR，它将 TSQR 用作适用于不太瘦的矩阵到方阵的面板分解）[7] 可实现快得多的实现：与豪斯霍尔德 QR 相比，TSQR 所需传输的数据量和“消息”（数据传输次数）在对数尺度上更少。这为当今基于缓存的多核CPU、高度并行的 GPU [1] 以及由其组成的集群 [22, 5]上的高性能 QR 实现奠定了基础。[23, 28]介绍了近期在 GPU 上为方阵实现 CAQR 的相关工作。其他与 TSQR 相关的工作包括：采用近似Householder反射的变体 [31], 、并行主元选取策略 [8], ，以及将 TSQR 与外部块修正Gram-Schmidt迭代相结合的方法，该方法无条件后向稳定 [2]。

一种避免通信的 TSQR 替代方法是基于格拉姆矩阵的方法，$C = X^T X$。这类方法固有的缺点是，在计算最终分解之前，输入矩阵 X 的条件数会被平方。另一方面，其核心计算是高瘦通用矩阵乘法，这是一种在 GPU 上经过充分优化的内核 [11]。SVQB [30], 实际上是一种高瘦奇异值分解，是最早从避免通信的角度发表的工作，尽管其基本原理早在几十年前就已为人所知。近来，Cholesky-QR 比 SVQB 得到了更多关注，可能是因为对于较大的 n，Cholesky分解的开销更低。为提高稳健性，该方法通过重新正交化执行两次 [3], ，我们将其称为 CholQR2 [34]。[30] 的作者也在“two-phase problem”（$QR = (I - YY^T)X$）的背景下考虑了重新正交化，即将 X 相对于某个已有的正交矩阵 Y 进行正交化。尽管从理论上讲，CholQR2 的开销是 TSQR 的两倍，但其实现可能效率更高， [16] 展示了在多核 CPU 上的实例。[35], 中通过以更高精度累加 Gram矩阵来缓解对条件数的限制；对于足够高瘦的矩阵，在多GPU系统上的性能与“标准”CholQR2 算法相近。为改善病态矩阵 X 的数值表现，[14] 提出了移位CholQR3 变体；当 $\kappa(X) < \mathcal{O}(\epsilon^{-0.5})$ 时，该方法具有后向稳定性。在 [13], 中比较了若干算法（HHQR、TSQR、移位CholQR3 和 CGS2）在 CPU集群上的性能，结果显示 CholQR3 具有明显优势。[15], 提出了一种带列主元选取的迭代CholQR，无需进行第二遍计算。

最后，随机数值线性代数中的方法被提出用于降低 QR分解中的选主元成本。随机列选主元QR（QRCR [9, 25, 10]）通过对输入矩阵进行随机采样来估算列范数。在 [26], 中，随机化被用于对 CholQR算法的Gram矩阵进行预处理，以处理高而非特别瘦的矩阵。

3 方法论

本节将简要介绍几种适用于当前任务的算法：Cholesky-QR2、SVQB2 和（Householder）TSQR。基于屋顶线性能模型 [33], ，我们为每种方法建立实际性能界限。屋顶线模型基于以下假设：（i）数值算法的执行速度受数据与执行单元之间最慢的路径，以及执行所需算术运算的浮点运算单元所限制；（ii）数据传输与计算可以完美重叠。若算法的算术强度（浮点运算次数/字节）为I，数据所在内存的带宽为 b，浮点运算单元的理论峰值性能为 R_{peak}，则其 ***Roofline*** 性能上限为

$$R_{\text{roof}} = \min(R_{\text{peak}}, I \cdot b). \tag{1}$$

我们可以通过定义机器平衡点 $M = \frac{R_{\text{peak}}}{b}$，确定特定算法/硬件组合的主要瓶颈。如果 $I < M$，该算法称为内存受限，否则称为计算受限。在“拐点” $M = I$附近，该模型通常并不现实。因此，我们使用 mixbench [21]微基准测试来确定 R_{roof}。该模型的局限性在于，未考虑将张量核心用于类GEMM运算（mixbench 使用 CUDA 核心执行融合乘加（FMA）），而且“完美重叠”的假设很容易在 GPU 上被打破，因为 GPU 缺乏现代 CPU 所具备的乱序执行能力，并且延迟相对较高，难以通过计算将其隐藏。

3.1 CholQR2 和 SVQB2

这些算法首先计算格拉姆矩阵 $C = X^T X$。由于这使它们不适用于 $\kappa_2(X) > \sqrt{\epsilon_{mach}}$），为提高稳健性，我们始终将算法应用两次。尽管如此，所得方法仅适用于病态程度中等的矩阵。算法 1 概述了一种不带额外稳定化步骤的“无 Q”CholQR2 算法。

算法 1无 Q 的 Cholesky-QR2

输入： $X \in \mathbb{R}^{m \times n}, m \gg n$
输出： $R : (XR^{-1})^T (XR^{-1}) = I$
1: $C \leftarrow X^T X$ *(tsmttsm, 设备)*
2: $R_1 \leftarrow$chol(C) *(主机)*
3: $C \leftarrow (X/R_1)^T (X/R_1)$ *(tsmRttsmR, 设备)*
4: $R_2 \leftarrow$chol(C)（主机）
5: $R \leftarrow R_2 \cdot R_1$

对于性能模型，我们将忽略不涉及 m 的步骤，因为我们关注的是列数极少的情况（ $n \lessapprox 100$）。

– tsmttsm 读取 $m \cdot n$ 个双精度数并执行 $2mn^2$ 次浮点运算，因此其计算强度为 $I_{tsmttsm} = \frac{n}{4}$ 浮点运算次数/字节。– tsmRttsmR 读取 $m \cdot n$ 个双精度数并执行 $3mn^2$ 次浮点运算，因此其算术强度为 $I_{tsmRttsmR} = \frac{3n}{8}$ 浮点运算次数/字节。– 仅涉及 $n \times n$ 矩阵的运算不予考虑。在实验中，对于较大的两组运行（见表3），这两个“高而窄”内核占总运行时间的 98.7% 以上。

该模型的一个局限是，它忽略了三角求解过程中线程块内的数据依赖。

对于 SVQB2算法，第 2 步和第 4 步中的 Cholesky分解被特征值分解 $C = U\Lambda U^T$ 所取代，而第 3 步中的三角求解则被与 $D = U\Lambda^{-1/2}$ 的矩阵-矩阵乘法所取代。这种方法本身就包含主元选取，因此有助于处理秩亏，或通过将 $\Lambda^{-1/2}$ 对角线上对应的元素替换为 0 来截断较小的奇异值。第 3 步的浮点运算次数从 $3mn^2$ 增加到 $4mn^2$，使算术强度达到 $I_{tsmmttsmm} = \frac{n}{2}$ 浮点运算次数/字节。除此之外，SVQB2 的开销相同：根据 Roofline 屋顶线模型，对于高而极瘦的矩阵，与 CholQR2 相比，其稳健性提升几乎无需付出额外代价。消除三角方程求解也有望提高性能模型的预测能力。

我们的算法实现基于 NVIDIA Warp [24]，它为共享内存中的矩阵乘积和三角方程求解提供了瓦片原语。我们使用固定数量的线程和网格步长循环，以减少归约期间的原子加法次数。

3.2 Householder TSQR

我们的实现分为两个阶段：首先，k 个线程块各自在共享内存中对 X 的一个包含 m/k 行的小块进行分解，并将所得的 R_i 写回全局内存，从而形成 $Y \in \mathbb{R}^{kn \times n}$。然后，一个线程块对 Y 进行分解，得到最终的三角因子 R，如算法 2 所示。

算法 2 在图形处理器上调用无Q Householder TSQR 内核

输入： $X \in \mathbb{R}^{m \times n}$，$m \gg n$，图形处理器线程块数量 k
输出： $R \in \mathbb{R}^{n \times n} : QR = X$，其中 Q 是某个正交矩阵

1: $\begin{pmatrix} Y_1 \\ \vdots \\ Y_k \end{pmatrix} \leftarrow \begin{pmatrix} Q_{1,1}^T & & \\ & \cdots & \\ & & Q_{1,k}^T \end{pmatrix} \begin{pmatrix} X_1 \\ \vdots \\ X_k \end{pmatrix}$ *(Qless_hhTSQR⟨k⟩, 设备)*

2: $R \leftarrow Q_2^T Y$ *(无Q_hhTSQR⟨1⟩, 设备)*

小型共享内存 hhTSQR 分解见算法 3 和 4。线程块数量 k 和内部块大小 b 的选择

算法 3 Qless_hhTSQR GPU内核（每个线程块）

输入： $X \in \mathbb{R}^{(m/k)\times n}$，$m >> n$，块大小 b
输出： $R \in \mathbb{R}^{n\times n}$：$QR = X$ 对于某个由正交列组成的矩阵 Q

1: $R \leftarrow 0$ （共享内存）
2: $W \leftarrow X_{1:b,\ :}$ （异步复制到共享内存）
3: **对于** $i = 1, \ldots, m/k/b$ **执行**
4: $\begin{pmatrix} W \\ R \end{pmatrix} \leftarrow Q_i^T \begin{pmatrix} W \\ R \end{pmatrix}$ *(原位梯形HHQR，覆盖 W的上半部分)*
5: **如果** $i + 1 < m/k/b$ **则**
6: $R \leftarrow W_{1:n,\ :}$ *(在共享内存中移动数据)*
7: $W \leftarrow X_{(i+1)b+1:(i+2)b,\ :}$ *(异步复制到共享内存)*
8: **结束如果**
9: **结束循环**
10: $R \leftarrow W_{1:n,:}$ *(复制到全局内存)*

算法4 原位梯形 HHQR 内核

输入： $\begin{pmatrix} W \\ R \end{pmatrix} \in \mathbb{R}^{(b+n)\times n}$

输出： $W'_{1:n,1:n} = Q^T \begin{pmatrix} W \\ R \end{pmatrix}$ 对于某个正交列矩阵 Q

1: **对于** $i = 1, 3, 5, \ldots, n$ **do**
2: 计算 $W_{i\quad ,i}$ 并为第 i 列计算豪斯霍尔德向量 v
3: 将 v 应用于第 $i + 1$ 列
4: 计算 $W_{i+1,\ i+1}$ 并为第 $i + 1$ 列计算豪斯霍尔德向量 w
5: 将 $(v,\ w)$ 按每次 4 列的步长应用于第 $i + 2$、.....、n 列。
6: **结束循环**

取决于硬件属性，其中最重要的是流式多处理器的数量以及每个线程块可用的共享内存容量。

对于性能模型，我们再次考虑 GPU 全局内存传输和双精度（融合乘加）浮点运算（不使用张量核心），并省略不涉及 m的项。算法 2 需要将完整矩阵 X 从全局内存传输到共享内存一次，以执行第一次（并行）无 Q 的_hhTSQR，即加载 mn个双精度数。与 Y 中 kn^2 个元素相关的内存流量不予考虑。FP64 浮点运算次数的下界为 $2mn^2$，其中我们只计算 m/b个梯形 HHQR 步骤，每步执行 $2(b + 1)n^2$ 次浮点运算。归约和第二次 Householder TSQR 调用中的某些运算（仅需 $O(n^3)$次运算）不予考虑。由此得到计算强度为 $I_{\mathrm{c,\ TSQR}} = \frac{n}{4}$ 浮点运算/字节。该模型的主要局限在于算法 4 第 5 行。这里是一个很小的乘积，由一个$2 \times b$ 和一个 $b \times 4$ 矩阵构成，前者位于寄存器中，后者位于共享内存中。要确定准确的性能上界，必须将共享内存延迟和带宽纳入考量，但这超出了本研究的范围。总之，我们的 TSQR 性能模型

预计在 m非常大时最为准确；尤其对于（接近）计算受限的情况，即 n较大时，模型会过于乐观。

4 数值实验

现在，我们已经可以对所提出的实现进行测试。为使讨论更加清晰，我们聚焦于一种 GPU 类型（NVIDIA H100）。如表 1 所示，目前市场上的高端 GPU 在本研究相关的指标方面具有定性上的相似性，即 HBM 带宽、共享内存大小和双精度浮点运算速率。对于内存带宽，我们不采用理论值，而是使用 $b = 2.15GB/s$；该值是通过 mixbench 测得的最大值。

GPU 型号	SM/CU 数量[1]	共享内存 每个 SM/CU	HBM 容量	HBM 带宽（向量）	FP64 FMA（张量）	FP64
NVIDIA H100	132	228 KB	80 GB	3.4 TB/s	34 TFlop/s	67 TFlop/s
NVIDIA B100	160	228 KB[2]	192 GB	8.0 TB/s	30 TFlop/s	40 TFlop/s
AMD MI300X	304	64 KB[3]	192 GB	5.3 TB/s	82 TFlop/s	163 TFlop/s
AMD MI350X	256	64 KB[3]	288 GB	8.0 TB/s	72 TFlop/s	144 TFlop/s

表 1. NVIDIA[4] 和 AMD[5]近期 GPU 的相关硬件特性。

4.1 实验设置

我们构造矩阵 $X \in \mathbb{R}^{m \times n}$，使得 $n = [1, 8, 16, 32, 64]$，同时保持 mn不变（即，保持 X 的内存占用不变，同时增加列数，从而提高计算强度）。条件数 $\kappa_2(X)$ 保持小于 $\epsilon_{DP}^{-0.5}$，以确保一次不选主元豪斯多尔夫 QR，或两次基于格拉姆矩阵的方法（如 CholQR2 或 SVQB2）能够得到稳定且准确的解。在运行基准测试之前，会先进行几次“预热”运行。这意味着 X 已驻留在设备上，不会将即时编译计入测量等。每种情况至少运行 50 次，并报告平均运行时间。

[1] 与 1000W 的 B200 相比，NVIDIA B100 的功耗通常受限于 700W。[2] NVIDIA Blackwell 为每个流式多处理器增设了独立的 256 KB 张量内存（TMEM），此外还有 228 KB 共享内存。本文未使用这部分内存。[3] AMD 使用计算单元（CUs）。在 CDNA 3/4 中，每个计算单元通常配备 64 KB 本地数据共享（LDS），相当于 AMD 版本的共享内存。[4] NVIDIA Corporation。（2025）。NVIDIA 数据中心技术网站。https://www.nvidia.com/en-us/data-center/h100/，以及 https://www.nvidia.com/en-us/data-center/technologies/blackwell-architecture/。[5] Advanced Micro Devices, Inc.（2025）。AMD Instinct 规格。https://www.amd.com/en/products/accelerators/instinct/mi300/mi300x.html，以及 https://www.amd.com/en/products/accelerators/instinct/mi350/mi350x.html。

4.2 豪斯霍尔德 QR 与厂商库 cuSOLVER

我们的首批实验旨在确认，在目标平台上，针对高而极瘦 QR分解，确实需要专用算法和实现。表 2 比较了针对三种不同问题规模 mn 的 CUDA（v13.1）不选主元豪斯霍尔德 QR 实现，以及 NVHPC v26.1 随附的 cuSOLVER 对 LAPACK dgetrf 的实现的运行时间。此处屋顶线模型的时间假设 X 从 HBM 读取一次，而 Q 写回 HBM。从理论上说，所有情况都受内存限制。

m	n	“标准”HHQR	cuSOLVER dgetrf	Roofline [毫秒]
$8.192 \cdot 10^6$	8	3.1	89.3	0.48
$4.096 \cdot 10^6$	16	14.6	90.9	0.48
$2.048 \cdot 10^6$	32	57.3	8.3	0.48
$1.024 \cdot 10^6$	64	214.6	9.9	0.48
$8.192 \cdot 10^7$	8	52.6	889	4.8
$4.096 \cdot 10^7$	16	105.2	917	4.8
$2.048 \cdot 10^7$	32	256.1	75.7	4.8
$1.024 \cdot 10^7$	64	1690	86.4	4.8
$8.192 \cdot 10^8$	8	545	–	48
$4.096 \cdot 10^8$	16	1098	–	48
$2.048 \cdot 10^8$	32	2267	752	48
$1.024 \cdot 10^8$	64	5700	853	48

表 2。标准豪斯霍尔德 QR（非 TSQR）在不进行主元选取时的运行时间（以毫秒为单位；采用自主实现和 cusolverDnXgeqrf）。两种例程都以隐式形式计算正交矩阵 Q。标记为“–”的情况由于 cuSOLVER 存在 32 位索引限制而无法运行。

从表 2 可以看出，对于极窄矩阵（$n \leq 16$），一种直接的实现方式可以优于厂商提供的库，但两种实现的运行时间都远未达到屋顶线模型的预测值。

4.3 CholQR2 和 SVQB2

基于格拉姆矩阵的方法的性能主要由 tsmttsm 内核（计算 $C = X^T X$）以及融合内核 tsmRttsmR（$C = (X/R)^T (X/R)$）或 tsmmttsmm（$C = (XB)^T (XB)$）决定。这些融合内核将高瘦矩阵乘法与小型三角求解（CholQR2）或小型矩阵乘法（SVQB2）相结合。如图 1 所示，后者达到的 Roofline 性能比例更高，因为它避免了按线程块顺序执行的分块三角求解。

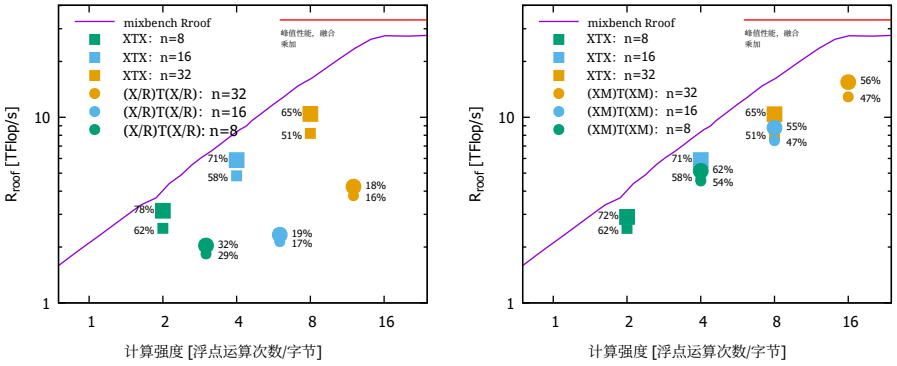

图1. CholQR2（左）和 SVQB2（右）中的两个 GPU 内核达到 Roofline 性能的百分比，随列数 n变化。行数 m 对应表3中最小（最大）的一组实验，分别以小（大）标记表示。

4.4 TSQR

无 Q 的 TSQR 只需加载一次矩阵 X，因此在纯内存受限区域中，理论上速度可达到 CholQR2 或 SVQB2 的两倍。另一方面，其内核并非基于通用矩阵乘法，因此要实现高性能更具挑战性。图 2 显示，我们的实现对于 $n \leq 8$ 达到了理想的 Roofline 性能。对于 $n > 8$，同步效应、指令

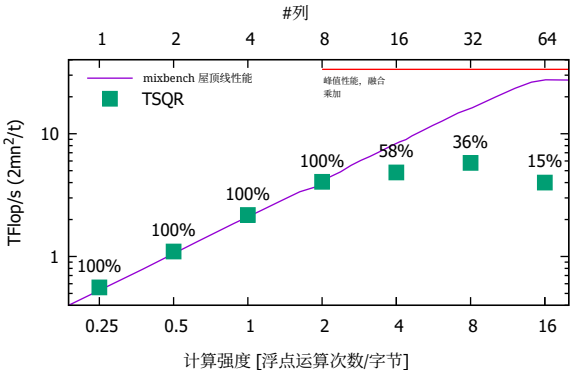

图 2. 我们的 TSQR 实现针对不同列数所达到的 Roofline 性能百分比 R_{roof} n。

传输延迟和共享内存传输将性能限制在约 6 TFlop/s（$n = 32$）。对于 $n > 32$，由于需要采用较小的块大小以适应共享内存（同步开销更大），性能会有所下降。

4.5 方法比较

表3列出了 H100 上 CholQR2、SVQB2 和 TSQR 的计时结果，以便直接比较。对于所研究的高而极窄的情形，标准的“全局”Householder-QR 以及 NVIDIA 在 cuSOLVER 中的实现（如表2所示）速度都很慢，很容易被更适合的算法超越。有趣的是，SVQB2 变体的性能几乎是 CholQR2 的两倍

		CholQR2		SVQB2		无 Q 的 TSQR
m	n	t_{mean}	t_{other}	t_{mean}	t_{other}	t_{mean}
$8.192 \cdot 10^6$	8	1.384	0.1111	1.051	0.1732	0.33
$4.096 \cdot 10^6$	16	2.021	0.1162	1.204	0.2097	0.62
$2.048 \cdot 10^6$	32	2.306	0.1291	1.484	0.3188	1.8
$1.024 \cdot 10^6$	64	-	-	-	-	10.8
$8.192 \cdot 10^7$	8	11.2	0.1448	7.924	0.2102	2.5
$4.096 \cdot 10^7$	16	17.39	0.1644	8.728	0.2503	4.5
$2.048 \cdot 10^7$	32	19.27	0.1783	9.957	0.352	8.1
$1.024 \cdot 10^7$	64	-	-	-	-	28.8
$8.192 \cdot 10^8$	8	110.9	0.3276	77.3	0.3921	25.8
$4.096 \cdot 10^8$	16	171	0.3444	83.83	0.4427	43.4
$2.048 \cdot 10^8$	32	189.2	0.357	95.19	0.5764	72.5
$1.024 \cdot 10^8$	64	-	-	-	-	209.7

表3. CholQR2 和 SVQB2 的运行时间（毫秒）。t_{other} 包括在 CPU 上进行小型矩阵分解所需的时间、Python 和内核启动引入的延迟等。

因为底层融合内核 tsmmttsmm 能利用比 tsmRttsmR 中三角方程求解更高的并行度。总体而言，SVQB2 通过相对直接的实现，达到了较高比例的 Roofline 性能（$\gtrsim 50\%$）。相比之下，我们精心设计的 TSQR 实现对于列数很少的情况（$n \leq 8$）可达到 100% 的 Roofline 性能，因此在此情形下至少比 SVQB2 快两倍。对于更多列（$n = 32$），SVQB2 的运行时间约为 TSQR 实现的 130%。如果采用更精心设计的实现，SVQB2 在这种情况下可能会胜过 TSQR，因为 TSQR 的内部并行度较低，因此难以实现高效执行。

4.6 超过 32 列

上文展示了列数最多为 32 时内存受限情形的实现和结果。这里的关键思路是利用图形处理器上的共享内存减少内存流量，并且不显式计算 Q矩阵（也不存储 Householder 反射子）。当列数较多时，这种方法的效果会打折扣，原因有二：首先，“无 Q”变体的优势

会随着所涉及的矩阵运算变得更加受计算限制而逐渐减小。图 3 展示了这一点：我们将无 Q 的 SVQB2 变体与一种简单的 CuPy 实现进行比较；后者会将 Q 存储一次（作为重新正交化步骤的输入），但在第二遍中不存储。从图 3 可以得出，CuPy

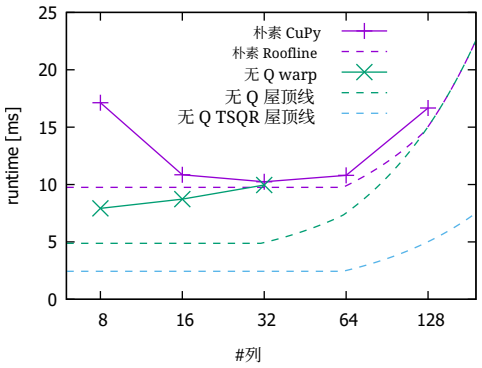

图 3. 使用 Warp 的无 Q SVQB2 实现与使用 CuPy 的“朴素”实现的比较。无 Q 变体的 Roofline 上限考虑了 $t_{X^TX}+t_{(XM)^T(XM)}$，而朴素变体考虑了 $t_{X^TX}+t_{XM}+t_{X^TX}$。从理论上讲，无 Q 的 TSQR 所需的内存传输量是其一半，浮点运算量是其三分之一。

对于 $n >= 32$，该实现达到了非常出色的 Roofline 性能。此外，对于更大的情况，底层 CuBLAS 运算可以在 H100 上使用张量核心，但此处未将其纳入考虑。

当前 GPU 上无 Q SVQB2 处理 $n > 32$ 时的第二个问题是，它严重依赖共享内存中的矩阵乘积。分块处理单个通用矩阵乘法（或本例中的对称秩-k 更新 $C = X^TX$）效果很好，但无法应用于融合内核所需的三重乘积 $(XB)^T(XB)$。

我们在 H100 上实现的 TSQR，利用给定的共享内存容量，最多可处理 64 列。从理论上讲，对于 $n > 64$，无 Q 的 TSQR 所需的运算次数会少于 SVQB2（以及 CholQR2）。但考虑到当前 GPU 的限制，我们找不到可行的实现方法。因此，对于更大的 n，分块 Gram–Schmidt 等非无 Q 算法变体就很有用。

5 总结与结论

我们基于 NVIDIA H100 GPU 的 Roofline 性能模型，分析了几种高瘦 QR 分解算法的性能。我们重点关注极高瘦的情形（$n \leq 64$ 列），此时问题本质上受内存限制。作为一项优化，我们避免显式或隐式存储矩阵 Q，以减少数据传输（“无 Q”QR）。结果表明，对于 $n < 32$ 列，这项优化带来了显著的加速。

更具体地说，我们测试了 TSQR、CholQR2（将 Cholesky-QR 应用两次）和 SVQB2（将 SVQB 应用两次）算法的无 Q 变体。从理论上讲，TSQR 所需的数据传输和运算最少。我们精心设计的 TSQR 实现速度最快；与 SVQB2 相比，其加速比从 3（对于 $n = 8$ 列）到 1.3（$n = 32$）不等。SVQB2 是

第二快的算法，而且仍然明显快于 CholQR2，因为后者需要进行三角方程求解，其并行度天生低于纯粹的矩阵乘矩阵乘法。SVQB2 和 CholQR2 的实现都使用内核融合，例如，一步完成 $(XB)^T(XB)$，从而减少数据传输。我们表明，当列数 $n \geq 64$ 较大时，这一点变得不那么重要，因为此时算法会变为计算受限，而直接实现的 SVQB2 也能提供良好的性能。

遗憾的是，TSQR 算法受限于图形处理器上的共享内存容量，实际上只能用于 $n \leq 32$ 列。总体而言，与供应商库 cuSOLVER 相比，我们在 $n = 32$ 列时实现了 10 倍的加速（在 $n = 8$时则为 $\gg 300$ ）。因此，我们认为有必要为高瘦矩阵开发更好的 QR 算法。在可移植性与性能之间权衡，SVQB2 似乎最有前景。

致谢。作者谨此感谢埃尔朗根国家高性能计算中心（NHR@FAU）（埃尔朗根-纽伦堡弗里德里希-亚历山大大学（FAU）下属机构）提供的科研支持和高性能计算资源。相关硬件由德国研究联合会（DFG）资助。

本研究部分得益于德国航空航天中心（DLR）的量子计算计划以及德国联邦研究、技术与航天部的支持，详见 qci.dlr.de/projects/QuTeNet。

利益披露。作者声明不存在与本文内容相关的任何竞争性利益。

参考文献

1. Anderson, M., Ballard, G., Demmel, J., Keutzer, K.：适用于 GPU 的通信规避 QR 分解。载于：2011 年 IEEE 国际并行与分布式处理研讨会。第 48–58 页。IEEE（2011 年 5 月）。https://doi.org/10 15 2 1109 2011 . /IPDPS. . . Barlow, J.L.：分块修正 Gram–Schmidt 算法及其分析。《SIAM 矩阵分析与应用期刊》40(4), 1257–1290（2019 年 1 月）。https://doi.org/10 1197400 3 1137 18. / m . Björck, Å.：《最小二乘问题的数值方法》。工业与应用数学学会（SIAM），费城，宾夕法尼亚州（1996 年 1 月）。https://doi.org/10 9781611971484 4 1137 1. / . . Chevalier, S., Parker, R.B.：面向基于图形处理器的线性求解器的扰动诱导静态主元选取。载于：2024 年 IEEE 电力与能源学会年会（PESGM）。第 1–5 页。IEEE（2024 年 7 月）。https://doi.org/10 10689118 5 1109 51994 2024 . /pesgm . . . Constantine, P.G., Gleich, D.F.：MapReduce 架构中的高瘦 QR 分解。载于：第二届 MapReduce 及其应用国际研讨会论文集。第 43–50 页。HPDC ’11，ACM（2011 年 6 月）。https://doi.org/10 1996103 6 1145 1996092 . / . . Cox, A.J., Higham, N.J.：加权最小二乘问题中 Householder QR 分解的稳定性。载于：Griffiths, D.F., Higham, D.J., Watson, G.A.（编）《数值分析 1997：第 17 届邓迪两年一度会议论文集》。

Pitman 数学研究札记，第 380 卷，第 57–83 页。朗文（1998），https://nhigham.com/wp-content/uploads/2023/08/cohi98.pdf 7. Demmel, J., Grigori, L., Hoemmen, M., Langou, J.：通信最优的并行和串行 QR 与 LU 分解。《SIAM 科学计算杂志》**34**(1), A206–A239（2012 年 1 月）。https://doi.org/10.1137/080731992 8. Demmel, J.W., Grigori, L., Gu, M., Xiang, H.：带列主元选取的通信规避秩揭示 QR 分解。《SIAM 矩阵分析与应用期刊》**36**(1), 55–89（2015 年 1 月）。https://doi.org/10.1137/13092157x 9. Duersch, J.A., Gu, M.：带列主元选取的随机 QR。《SIAM 科学计算杂志》**39**(4), C263–C291（2017 年 1 月）。https://doi.org/10.1137/15m1044680 10. Duersch, J.A., Gu, M.：用于秩揭示矩阵分解和低秩近似的随机投影。《SIAM 综述》**62**(3), 661–682（2020 年 1 月）。https://doi.org/10.1137/20m1335571 11. Ernst, D., Hager, G., Thies, J., Wellein, G.：面向 GPU 上实数和复数高瘦矩阵乘法内核的性能优化。《国际高性能计算应用杂志》**35**(1), 5–19（2021）。https://doi.org/10.1177/1094342020965661, https://doi.org/10.1177/1094342020965661 12. Fan, H.Y., Zhang, L., w. Chu, E., Wei, Y.：内积空间中的无 Q 的 QR 分解。《线性代数及其应用》**491**, 292–316（2016 年 2 月）。https://doi.org/10.1016/j.laa.2015.08.035 13. Fukaya, T.：《分布式并行高而瘦 QR 分解：各种系统上多种算法的性能评估》，第 275–287 页。施普林格·自然瑞士（2023）。https://doi.org/10.1007/978-3-031-29927-8_2214. Fukaya, T., Kannan, R., Nakatsukasa, Y., Yamamoto, Y., Yanagisawa, Y.：用于计算病态矩阵 QR 分解的移位 Cholesky QR。《SIAM 科学计算杂志》**42**(1), A477–A503（2020 年 1 月）。https://doi.org/10.1137/18m1218212 15. Fukaya, T., Nakatsukasa, Y., Yamamoto, Y.：一种用于计算带列主元选取的高瘦 QR 分解的 Cholesky QR 类算法。载于：2024 年 IEEE 国际并行与分布式处理研讨会（IPDPS）。第 63–75 页。IEEE（2024 年 5 月）。https://doi.org/10.1109/ipdps57955.2024.00015 16. Fukaya, T., Nakatsukasa, Y., Yanagisawa, Y., Yamamoto, Y.：CholeskyQR2：一种用于在大规模并行系统上计算高瘦 QR 分解的简单通信规避算法。载于：2014 年第五届大规模系统可扩展算法最新进展研讨会。第 31–38 页。IEEE（2014 年 11 月）。https://doi.org/10.1109/ScalA.2014.11 17. Golub, G.H., Van Loan, C.F.：《矩阵计算》。约翰斯·霍普金斯大学出版社，巴尔的摩，马里兰州，第 4 版（2013）。https://doi.org/10.56021/9781421407944 18. Higham, N.J.：半正定矩阵 Cholesky 分解的分析，第 161–185 页。牛津科学出版物，克拉伦登出版社，牛津（1990）19. Higham, N.J.：《数值算法的精度与稳定性》。应用数学其他著作第 80 号，工业与应用数学学会，费城，宾夕法尼亚州，第 2 版（2002 年 1 月）20. Kawamura, H., Suda, R.：使用带主元选取的 QR 分解的低秩矩阵近似算法的截断误差最小上界。《日本工业与应用数学杂志》**38**(3), 757–779（2021 年 2 月）。https://doi.org/10.1007/s13160-021-00459-x

21. Konstantinidis, E., Cotronis：利用微基准测试和硬件指标分析，构建用于估算 GPU 内核性能的定量屋顶线模型。《并行与分布式计算杂志》**107**，37–56（2017年9月）。https://doi.org/10.1016/j.jpdc.2017.04.002 22. Langou, J.：使用 MPI_Reduce 计算高瘦矩阵 QR分解的 R（2010年2月）。https://doi.org/10.48550/ARXIV.1002.4250 23. Leng, Y., Zou, G., Wang, H., Wu, P., Zhang, S.：在新兴 GPU 架构上利用张量核心实现高性能 Householder QR分解。《IEEE并行与分布式系统汇刊》**36**(3)，422–436（2025年3月）。https://doi.org/10.1109/tpds.2024.3522776 24. Macklin, M.：Warp：用于GPU模拟和图形学的高性能 Python框架（2022年3月），https://github.com/NVIDIA/warp，NVIDIA GPU 技术大会（GTC）25. Martinsson, P.G., Quintana OrtÍ, G., Heavner, N., van de Geijn, R.：结合随机化进行列主元选取的 Householder QR分解（HQRRP）。SIAM 科学计算杂志 **39**(2)，C96–C115（2017年1月）。https://doi.org/10.1137/16m1081270 26. Melnichenko, M., Balabanov, O., Murray, R., Demmel, J., Mahoney, M.W., Luszczek, P.：针对高矩阵的随机化和主元选取 CholeskyQR（CQRRPT）。SIAM 矩阵分析与应用杂志 **46**(3)，1701–1734（2025年7月）。https://doi.org/10.1137/24M163712X，https://doi.org/10.1137/24M163712X 27. Röhrig-Zöllner, M., Thies, J., Basermann, A.：现代多核 CPU 上大型稠密张量低秩 TT-SVD 的性能。《SIAM 科学计算杂志》**44**(4)，C287–C309（2022年7月）。https://doi.org/10.1137/21M1395545 28. Schofield, I., Alimohammad, A.：利用 GPU 并行化 QR分解。载于：2019年IEEE加拿大电气与计算机工程会议（CCECE），第1–6页。IEEE（2019年5月）。https://doi.org/10.1109/ccece.2019.8861519 29. Sid-Lakhdar, W., Cayrols, S., Bielich, D., Abdelfattah, A., Luszczek, P., Gates, M., Tomov, S., Johansen, H., Williams-Young, D., Davis, T., Dongarra, J., Anzt, H.：PAQR：避免主元选取的QR分解。载于：2023年IEEE国际并行与分布式处理研讨会（IPDPS），第322–332页。IEEE（2023年5月）。https://doi.org/10.1109/ipdps54959.2023.00040 30. Stathopoulos, A., Wu, K.：一种同步需求恒定的分块正交化过程。《SIAM 科学计算杂志》**23**(6)，2165–2182（2002年1月）。https://doi.org/10.1137/s1064827500370883 31. Tomás, A.E., Quintana-Ortí, E.S.：在图形处理器上使用近似 Householder 反射子的高而窄 QR分解。《超级计算杂志》**76**(11)，8771–8786（2020年1月）。https://doi.org/10.1007/s11227-020-03176-3 32. Trefethen, L.N., Bau, III, D.：《数值线性代数》。工业与应用数学学会，费城，宾夕法尼亚州（1997）。https://doi.org/10.1137/1.9780898719574 33. Williams, S., Waterman, A., Patterson, D.：Roofline：一种直观的多核架构性能可视化模型。《美国计算机协会通讯》**52**(4)，65–76（2009年4月）。https://doi.org/10.1145/1498765.1498785 34. Yamamoto, Y., Nakatsukasa, Y., Yanagisawa, Y., Fukaya, T.：Cholesky QR2算法的舍入误差分析。《电子数值分析汇刊》**44**，306–326（2015），https://etna.ricam.oeaw.ac.at/volumes/2011-2020/vol44/abstract.php?pages=306-326

35. Yamazaki, I., Tomov, S., Dongarra, J.：混合精度 Cholesky QR分解及其在配备多块 GPU 的多核 CPU 上的案例研究。《SIAM 科学计算杂志》**37**(3)，C307–C330（2015年1月）。https://doi.org/10.1137/14m0973773