arXiv:2611.30275v1 [cs.SC] 23 Nov 2026

并行化复矩阵乘法：3M 方案

Yuval Spiizer特拉维夫大学
yuval.lahav@mail.huji.ac.il

摘要。 我们研究两个复数矩阵相乘的问题。这是量子化学、电磁仿真、量子电路仿真以及基于 FFT 的方法中使用的核心运算。标准做法早已采用经典的 $3M$（高斯/卡拉楚巴）恒等式来重构局部计算：通过三次实矩阵乘法而非四次来计算乘积，无条件地减少 25 % 的算术运算量，但通信量保持不变。我们证明，是否值得采用 3M，以及是否还值得通过在不相交的处理器组上并行计算三个乘积来重构通信、进一步发挥 3M 的优势，是两个彼此独立的问题；并在标准 α-β-γ 通信模型下，分别给出闭式解答。在该模型下，对于二维布局，在大-P 极限下，并行化这一划分可使带宽减少 13.4%，延迟减少 $1-1/\sqrt{3}\approx 42.3\%$；后者与底层算法的算术指数无关。三维布局则无法实现这两种减少。对于 Strassen 算法，算术方面的情况恰恰相反：局部重构会更差7.1%，而非更好。我们还将结果推广到 Toom-Cook-k，给出一个随 k 增大而变化的闭式条件；满足该条件时，高阶划分可以完全弥补三维布局的不足。例如，$k=3$ 就足以使三维布局对经典矩阵乘法有利，而常规的$k=2$ 划分则始终无法做到这一点。这些数值是闭式模型的预测，并非来自分布式实现的测量结果；我们在第 9 节讨论该模型忽略了哪些因素，以及校准模型需要做些什么。

关键词。 并行矩阵乘法、复数矩阵乘法、通信规避算法、Toom-Cook 算法、Strassen 算法、分布式内存计算

1 引言 复数矩阵乘法是量子化学 [49, 55],电磁仿真 [27, 50], 量子电路仿真 [37, 3], 以及任何处理复数数据的 FFT 方法中的核心内核 [19]。

随着矩阵维度增大，计算这样的乘积 $C=A\times B$ 需要将 A、B 和 C分配到 P 个处理器上。这引出了一个普遍的问题：并行矩阵乘法，在过去四十年中已得到广泛研究 [15, 56, 17, 12, 22, 1, 38, 47, 32, 2, 30, 54, 20, 6]: 如何布局矩阵，以及如何安排必要的算术运算和数据移动，才能使分布式内存机器上的总运行时间最短。

有三个量决定了运行时间。算术成本 是每个处理器执行的标量乘法和加法次数，统计范围为 Level 3 BLAS 等库所依赖的稠密矩阵内核 [24]。通信成本，也称带宽，是每个处理器发送和接收的字数。延迟 是交换的消息数；它不取决于消息大小，因为每条消息无论包含多少数据，都会产生固定开销。BSP [54]和 LogP [20]等成本模型都基于这种区分。并行算法的总运行时间同时受这三个量支配，降低其中一项可能会增加另一项。例如，在更多处理器之间复制数据可以减少通信，但代价是增加内存占用和冗余算术运算。本文研究并行计算复矩阵乘积时这三项成本如何相互影响。

降低通信开销并非矩阵乘法所独有的关注点。数十年来，这一直是数值线性代数领域的核心目标，研究人员已为 QR 和 LU 分解 [23, 26], 、特征值与奇异值问题 [4], 以及 Krylov 子空间方法 [29]开发了通信最优算法。这些研究成果均已建立相应的通信下界 [6]。Toom-Cook [51,18](§2.4) 将 Karatsuba 推广到 k路拆分。Nissim 等人提出了 Toom-Cook-k 算法，可同时优化算术运算和输入/输出 [40]。随后，Nissim 和 Spiizer 将这项技术应用于内积 Toom-Cook，在算术运算与通信成本之间实现了权衡 [43]。Bilardi 和 De Stefani 为 Toom-Cook 整数乘法的输入/输出复杂度建立了紧确下界 [9], ，而 De Stefani 则分析了串行与并行混合整数乘法算法的输入/输出复杂度 [21], ，其并行情形

版权 © 20XX 归 SIAM 所有。未经授权不得复制本文

也已在容错 Toom-Cook 文献中单独进行分析 [42, 41]。

全文中，我们比较并行化 3M 恒等式的三种策略。它们将在下文 §1.2 首次完整命名，并在 §3 中正式定义：**V1**，我们提出的拆分为并发处理器组的方法；**V2**，一种保留标准通信调度的原地重构；以及 **V3**，标准的四次乘法方法。表 1.1 概述了由此得出的比较结果。

1.1 前人工作。 计算复数乘积的常见方法是逐项应用平凡公式 (1.1) $(a+bi)(c+di) = (ac-bd) + (ad+bc)i$

每个元素需要进行四次实数乘法和两次实数加法。经典算法的另一种方法只需三次乘法：高斯恒等式用于两个高斯整数相乘，也是 Karatsuba 算法所依据的同一恒等式 [33]。它可直接从标量推广到矩阵块 A_0, A_1, B_0, B_1，从而得到 $3M$ 恒等式

$$\begin{aligned} Q_1 &= A_0 B_0, \quad Q_2 = A_1 B_1, \\ Q_3 &= (A_0 + A_1)(B_0 + B_1), \\ \Re(C) &= Q_1 - Q_2, \\ \Im(C) &= Q_3 - Q_1 - Q_2, \end{aligned} \tag{1.2}$$

代价是每个元素需要进行四次实数加法。因此，逐元素来看，4M 更可取，因为它所需的总运算次数更少；只有当Q_1、Q_2、Q_3 本身是大型矩阵乘法时，3M 的优势才会显现，因为它省去的是一次完整的 $O(n^3)$ 运算，而非一次标量运算。

这种 3M/4M 权衡在单节点层面已得到充分研究。Van Zee 等人将两种方案都集成到 BLIS 中，只对局部原地计算 [57,58]进行重构；Kouya 将其扩展到任意精度运算 [35]；这种恒等式数值稳定性略差的问题也已被单独刻画 [28]。在 Ozaki 方案中也出现了结构完全相同的归约：该方案借助 NVIDIA 的 cuBLAS 等厂商库 [44], 所提供的张量核心硬件 [45, 39]„ 在低精度（FP8，8 位浮点数；INT8，8 位整数）下模拟 FP64（64 位浮点数）矩阵乘法；Uchino 等人最近提出了相同的 4 次运算减至 3 次的方案 [52]。这两条研究路线都未考虑分布式内存环境。

在分布式计算领域，通信规避矩阵乘法可追溯到 Hong 和 Kung 提出的输入/输出（I/O）下界 [30], , Aggarwal 等人 [2]和 Irony 等人 [32] 将其推广到分布式内存中，Ballard 等人（BDHS）对此进行了综述 [6]。§2.2 中的二维、2.5维和三维算法在不同内存预算下实现了这一界限 [15, 56, 17, 12, 22, 1, 38, 47], , 并采用 BSP 和 LogP 成本模型 [54, 20]。在实践中，内存受限时使用二维算法，因为它在每个处理器上只存储每个矩阵的一个副本；内存充足时使用三维算法，通过将输入复制 [30], 倍，大幅减少通信；而当可用内存足以容纳部分复制时，2.5维算法则在两者之间进行折衷。Strassen 算法及其后续算法有通信最优的分布式实现（CAPS，§2.5）[48, 59, 34, 8, 7, 36, 5]。

我们区分的是内层实数矩阵乘法原语与叠加于其上的外层3M（或 Toom-Cook-k）恒等式。上述通信规避和通信最优领域的研究 [7, 36, 5, 6], （包括 CAPS）针对的是原语本身，无论输入是实数还是 Strassen 值；这些研究并未探讨外层恒等式调用该原语三次（或 $2k-1$ 次）时会发生什么，因为它们并未涉及复数/高斯复合。关于 3M/4M [57, 52]的单节点研究确实考虑了外层恒等式，但始终只重构其局部计算（按我们的术语即 V2），并默认内层原语自身的通信调度固定且共享；这些研究没有考虑围绕外层恒等式本身重构通信，也就是我们的 V1。据我们所知，先前没有研究分析这一外层层面的问题，而这正是本文所填补的空白。

1.2 我们的贡献。

1.2.1 我们的研究成果。 我们的主要贡献是在标准 α-β-γ 通信模型下，对 V1、V2 和 V3 之间的三方比较（§1）给出闭式分解；下文所有百分比均为该模型的预测值，而非测量值；对于模型遗漏的内容以及校准该模型所需的工作，我们将在 §9 中再行讨论。对于在二维矩阵乘法场景下计算两个复矩阵的乘积，并将其与标准实践进行比较，该模型预测带宽渐近降低 13.4%，延迟降低 **42.3%**（推论 4.6）。在任何实际可达到的规模下，这一优势可扩展至临界复制因子 $c^* = P_{\log_3(4/3)} \approx P^{0.262}$（推论 4.8），即从二维复制到完整三维复制距离的大约 79%；超过该临界值后，原地重构（V2）将成为更优选择；而在天文数字般巨大、实际上无法达到的 P 下，拆分在整个 2.5维范围内（除完整三维复制本身之外）始终更具优势（推论 4.9）。

我们的第二项成果涉及 Strassen 算法。在二维情况下，拆分（V1）仍能减少通信，但代价是算术运算增加：在任何三通道拆分下，Strassen 重组阶段的开销都会增加三倍，因此原地重构（V2）在算术运算方面比标准方案（V3）高**7.1%**，这是 V1 也会继承的惩罚。从规模化角度看，通信节省仍占优势：带宽降低 3.7%，延迟则与经典算法一样降低 42.3%，因为延迟降低不依赖于算术指数，而 Strassen 的适用窗口更窄（临界复制因子 , 约为经典算法的三分之一）；超过这一范围后，

版权 © 20XX，SIAM。未经授权不得复制本文。

表 1.1：算术运算（V2 与 V3）和通信（V1 与 V2）。

设置	V2 与 V3（算术运算）	V1:V2 带宽（$P \to \infty$）	V1:V2 延迟（$P \to \infty$）
经典算法，二维	-25% [57]	-13.4%	-42.3%
经典算法，三维	-25% [57]	$+4.0\%$	$\approx 0\%$
Strassen ，2D	$+7.1\%$	-3.7%	-42.3%
Strassen ，3D	$+7.1\%$	$+11.6\%$	$\approx 0\%$
Karatsuba 分割	-25%	-13.4%	-42.3%

$k = 2$、 $n = 2000$。负值表示 $=$ 归约，正值表示 $=$ 增加。带宽和延迟是各项的渐近（$P \to \infty$）V1 相对于 V2 的比率，取自定理 4.4 和引理 4.5。预览：这一分解及其背后的两个定理将在 §4 中推导。

以及在三维情况下，标准方案（V3）在我们评估的所有规模下都是三种策略中最佳的选择。

我们将这一技术推广到多精度矩阵乘法：将其应用于 Karatsuba 的高位/低位数字分割，以便在低精度硬件上模拟高精度运算时，我们的定理无需修改即可沿用，并能正确预测出类似经典算法（而非类似 Strassen）的排序。进一步推广到 k 位数字分割（Toom-Cook-k），同样的优势会随着 k 增长；与两位数字的情况不同，这种方法还能完全挽救三维布局。

还有两个结果进一步完善了整体图景。拆分成更多部分可以挽救三维布局：对于所有算术指数，无论是经典算法还是 Strassen，Toom-Cook-3 在渐近意义上都足够；不过，Strassen 的交叉阈值需要更大的 P，因为仅带宽项在 $k = 3$ 时仍然有利于标准做法。对于任何类似的拆分技术，同样的阈值也适用；对于 Strassen 自身递归步骤这类缩减规模的算法，阈值取决于内存而非通信，这也解释了 CAPS 为何选择广度优先搜索/深度优先搜索。我们还检验了 V2 的额外乘积是否迫使它使用比 V3 更多的本地内存：事实并非如此。通过谨慎选择缓冲区复用方式，V2 可以使其内存占用与 V3 完全相同，而且只需调用现有的实矩阵乘法例程这一常规操作。接下来我们将说明如何得出所有这些结果。

1.2.2 我们如何得出这些结果。一旦A_0， A_1， B_0，B_1 被分布到 P 个处理器上（二位、2.5维或三维布局，§2.2），恒等式 (1.2) 就有不止一种并行化方式，而哪种方式最佳并不显而易见。我们提出通过将处理器拆分为并发组来并行化 3M（下文的 V1），并将其与另外两种现有方案（V2、V3）进行比较。具体而言，有三种可行策略：

V1 3M-Split（我们的方案）。将 P 个处理器划分为三个各含 $P/3$个处理器的组；每组并行计算 Q_1、 Q_2、 Q_3 中的一个。

V2 原位 3M。保持标准通信调度；仅对本地原地计算应用 3M 分解（如 BLIS 的 3m 内核 [57]）。

V3 标准。 上述常见方法：采用与 V2 相同的调度，并通过直接四乘法公式进行局部计算。

上述五项结果都源于将三方比较分解为两个相互独立的子问题，而每个子问题都可由一个定理单独回答。**V2 与 V3**是一个纯粹的算术运算问题，因为原地重构（V2）与标准方案（V3）按构造共享相同的通信调度（命题 4.1），因此二者的比较归结为比较两个首项系数， $c_1^{\mathrm{TC}}(k) = 2k - 1$ 与 $c_2(k) = k^2$（§2.3 和 §2.4），而与 n、 P 或通信模型无关。**V1 与 V2**是一个通信问题，因为拆分为 $2k - 1$ 个并发处理器组（V1）与原地重构（V2）依据浮点运算时间不变性论证（定理 4.3）共享相同的计算时间，因此二者的比较归结为带宽和延迟如何随处理器拆分而扩展。$\theta^*(k)$仅决定带宽项，从而决定 V1 随着 P 增长而追赶 V2 的速度；最终的胜者，如 $P \to \infty$ 所示，则由延迟决定。对于二维情况，无论 $\theta^*(k)$ 如何，延迟都会为 V1 带来一种通用且与算法无关的优势（推论 4.6）。表 1.1 将这两个分解后的问题具体化：3M 是否值得实施（现有实践已经回答了这一问题），以及是否还值得进一步拆分为组（本文所回答的问题）。这项分析基于模型；我们还提供了一个一致性检查模拟器作为补充材料[1]，，它在相同成本模型下，通过对每种布局的通信模式进行独立的离散核算，将我们的闭式预测与之交叉核对，从而发现闭式中的代数错误或常数偏差；但它不能替代真实分布式实现上的测量（§6、§9）。

2 预备知识

[1]匿名代码仓库：https://anonymous.4open. science/r/3M_mat_mul-7E4C。

版权 © 20XX，SIAM。未经授权不得复制本文

2.1 符号与模型。 如果某算法的乘法次数和加法次数满足 $M(n)\ A(n) = \Theta(n^\omega)$，则称其算术指数为 ω，其首项系数为 c_M、c_A；我们将这对参数（M、A）称为其算术特征。本文中的每种算法都具有 $\omega = 3$（经典算法）、$\omega_0 = \log_2 7 \approx 2.8074$（Strassen）或依赖于 k 的指数（Toom-Cook-k）。在通信方面，$BW(n、P) = n^2/P^\theta$（带宽，每个处理器的字数），而$S(n、P)$（延迟，每个处理器的消息数），均忽略低阶项。

我们将机器建模为由点对点通信网络连接的 P 个相同处理器，即标准的分布式内存模型 [25, 54, 20, 15, 56, 17, 12, 22, 1, 38, 47, 32, 2, 30, 42, 41]。我们采用通信规避文献中始终使用的双线性算法形式化 [30, 32, 7, 6, 40]。$C = A \times B$的计算方式为 $\text{vec}(C) = W\big((U^\top \text{vec}(A)) \odot (V^\top \text{vec}(B))\big)$，其中编码矩阵为 U，V，解码矩阵为 W [34, 40]，它们应用于子块（Strassen）或求值点（Toom-Cook）。运行时间为 $T = \gamma(M + A) + \beta BW + \alpha S$，其中机器常数为 γ，β，α。我们的目标是降低首项系数 c_M，c_A 以及带宽/延迟系数，而不仅仅是降低指数。一旦算法和布局选定，这些指数便固定不变 [13, 34, 8, 40]。

将布局的内存界代入 BDHS 界 [7],(2.1) $BW(n, P, M) = \Theta(n^\omega/(PM^{\omega/2-1}))$，即可得到本文中使用的每个带宽指数 $\theta(\omega)$。

2.2 二维、2.5维和三维矩阵乘法。这三种布局执行相同的总工作量n^3，区别仅在于内存预算 M，因此通信量也不同。它们涵盖了两个极端情况及其间的所有情况。二维（Cannon [15], SUMMA [56, 17, 12]）是有限内存极端：P 个处理器组成一个 $\sqrt{P} \times \sqrt{P}$网格，内存最少$M = \Theta(n^2/P)$，进行 $\sqrt{P}$ 轮局部乘加和移位/广播。三维（Dekel等人 [22], 改进自 [1, 38, 47]）是无限内存极端：复制 $P^{1/3}$倍，内存最多 $M = \Theta(n^2/P^{2/3})$，在 $P^{2/3}$层中的每一层上执行局部二维乘法，最后进行归约。由此可得 BDHS 界中的 $\theta_{2D}(3) = \frac{1}{2}$、$\theta_{3D}(3) = \frac{2}{3}$，与已知的通信最优结果 [32, 47]一致。2.5维 [47]通过复制因子 $c \in [1, P^{1/3}]$进行插值，$M = \Theta(cn^2/P)$，表示两个极端之间可用的任意内存；$c = 1$ 和 $c = P^{1/3}$ 则分别精确还原二维和三维。将此 M 代入 BDHS 界（2.1），可得经典矩阵乘法的 $BW(n, P, c) = \Theta(n^2/\sqrt{cP})$，并正确还原两个端点：$n^2/\sqrt{P}$在 $c = 1$处，以及 $n^2/P^{2/3}$在 $c = P^{1/3}$处。表 2.1 给出了三个方案的主导系数；复制只会改变通信量，不会改变 c_M、c_A。

表 2.1：经典矩阵乘法的通信特征。

布局	$BW(n, P)$	$S(n, P)$
二维（受限于 M）	$n^2/\sqrt{P}$	$\Theta(\sqrt{P})$
2.5维（c）	$\Theta(n^2/\sqrt{cP})$	$\Theta(\sqrt{P/c^3} + \log c)$
三维（无限 M）	$n^2/P^{2/3}$	$\Theta(\log P)$

$\omega = 3$；$c_M = c_A = 1$ 始终不变，只有 BW，S 会变化。为保持一致性，这里根据 BDHS 界（2.1） [15, 56, 22, 1, 47], 重新推导已知结果。

2.3 4M 和 3M 高斯乘积。对于实数A_0、A_1、B_0、B_1，高斯乘积 $C = (A_0 + iA_1)(B_0 + iB_1)$ 有两种形式。4M：常见方法（式（1.1））是在一次矩阵乘法遍历中，对每个元素逐一应用平凡的复数乘法公式。每个 n^3 标量乘加步骤都会计算一个复数乘积（4 次实数乘法、2 次实数加法），并将其累加到当前复数和中（再进行 2 次实数加法）：每步进行 4 次乘法和 4 次加法，因此得到$M_{4M}(n) = A_{4M}(n) = 4n^3 + O(n^2)$，即 $c_M = c_A = 4$。3M（高斯/卡拉楚巴）：对于标量 a、b、c、d，同一乘积也可以写作 $q_1 = ac$、$q_2 = bd$、$q_3 = (a+b)(c+d)$，从而得到 $\Re = q_1 - q_2$ 和 $\Im = q_3 - q_1 - q_2$，只需三次实数乘法，而非四次。无论 a、b、c、d是标量还是矩阵块，同样的分配展开都成立，因此该恒等式可直接推广到 A_0、A_1、B_0、B_1（式（1.2））：进行三次乘积运算，另加 $O(n^2)$ 的开销，得到 $M_{3M}(n) = A_{3M}(n) = 3n^3 + O(n^2)$，即 $c_M = c_A = 3$。

2.4 Toom-Cook-k。 Toom-Cook-k [51, 18]将 Karatsuba（$k = 2$）推广到 k 路拆分，最初用于长整数乘法：将每个输入写成关于 k 位数字块的 $k-1$ 次多项式，$A'(x) = \sum_{i=0}^{k-1} A^{(i)} x^i$；精确乘积是一个长度为 $2k-1$ 的卷积。普通乘法算法逐项相乘两个 k 位整数，代价为 $\Theta(k^2)$ 次单数字乘法（相比之下，将两个 k 位整数相加只需 $\Theta(k)$ 次运算）。Toom-Cook 则在 $2k-1$ 个点处对两个输入求值，逐点相乘（即缩小规模后的求值结果之间进行 $2k-1$ 次乘法），然后进行插值；求值和插值步骤合计使用 $O(k^2)$ 次加法。Karatsuba 是 $k = 2$（用 3 次乘法代替 4 次）；Toom-3 是 $k = 3$（用 5 次代替 9 次）；Toom-4 是 $k = 4$（用 7 次代替 16 次）。

2.5 Strassen 与 CAPS。 Strassen [48]使用分块矩阵进行乘法，每层递归乘法次数为 7 次而非 8 次，并进行 q 次矩阵加法（编码 + 解码），得到 $M(n) = 7M(n/2)$、$A(n) = 7A(n/2) + \frac{q}{4}n^2$。求解（通过第 5 节中的递归模拟验证）得到 $\omega_0 = \log_2 7$, $c_M = 1$, $c_A = q/3$。三种

版权 © 20XX 归 SIAM 所有，未经授权不得复制本文

算法 3.1V1：3M-Split

要求： A_0，A_1，B_0，B_1 分布在 P 个处理器上；组 ID $g \in \{1,2,3\}$

1: 将 P 个处理器划分为 3 个互不相交的组，每组大小为大小 $P/3$

2: 将 (A_0，B_0) 散布到组 1，将 (A_1，B_1) 散布到组 2，(A_0+A_1，B_0+B_1) 到组 3

3: **并发执行：** 每个组 g 计算 $Q_g \leftarrow$ 使用其实数矩阵乘法(·，·)处理器 $P/3$

4: 收集 Q_1，Q_2，Q_3；合并 $\Re(C) = Q_1 - Q_2$，$\Im(C) = Q_3 - Q_1 - Q_2$

变体在全文中均有使用（表 5.1）：经典算法，$q = 18$ [48]；Winograd，$q = 15$ [59]；Karstadt-Schwartz，$q = 12$ [34]，拆分为 6+6 编码/解码。这得到$\theta_{2D}(\omega_0) \approx 0.5963$、$\theta_{3D}(\omega_0) \approx 0.7309$，二者均大于经典算法的对应值，因为 $\omega_0 < 3$ 使 θ 趋近于 $\omega \to 2$处的 $\theta \to 1$ 极限。

CAPS [36, 5] 通过将 Strassen 的 7 路划分嵌套到 2.5D 布局中，在分布式内存上实现 Strassen：在回退到经典算法 2.5D 之前，先执行内存 M 允许的尽可能多层 Strassen，并在处理器之间进行复制。其算术特征恰为 $c_M = 1$，$c_A = q/3$ 无论 P还是复制都如此（如 §2.2 所述，只有通信会发生变化）；已有证明表明，它是通信最优的，在 $\omega = \omega_0$ 处达到 BDHS 界，适用于与 2.5D 相同的 $\theta_{2D}(\omega_0)$ 至 $\theta_{3D}(\omega_0)$ 系列。§5 中我们直接使用这一 $\theta(\omega_0)$ 系列，而不重新推导 CAPS 内部实现中的常数。

3 算法 现在，我们正式定义 V1、V2、V3（在 §1 中引入），并证明它们都能正确计算 $C = A \times B$。§4 将分析它们的算术成本、通信成本和内存占用。

3.1 V1：3M-Split（我们的方案）。算法 3.1 将 P处理器划分为三个互不相交的 $P/3$组；每个组分别计算 Q_1、Q_2、Q_3，并发地使用实数算法，所需处理器数量更少。

引理3.1（V1 正确）。算法 *3.1* 正确计算出 $C = A \times B$。

证明。通过构造，组 1 计算 $Q_1 = A_0B_0$，组 2 计算 $Q_2 = A_1B_1$，组 3 计算 $Q_3 = (A_0+A_1)(B_0+B_1)$，每组仅使用散布到其中的操作数块。展开 $Q_3 - Q_1 - Q_2 = (A_0+A_1)(B_0+B_1) - A_0B_0 - A_1B_1 = A_0B_1 + A_1B_0$ 可知，$\Re(C) = Q_1 - Q_2 = A_0B_0 - A_1B_1$ 和 $\Im(C) = Q_3 - Q_1 - Q_2 = A_0B_1 + A_1B_0$ 与复数乘法的直接定义完全一致。将处理器划分为互不相交的组不会影响这一恒等式，因为每个 Q_g 只依赖于自身的两个操作数块，而这两个块已全部提供给

算法 3.2 V2：原地 3M

要求： 局部块 A_0、A_1、B_0、B_1，以相同方式驻留在 P 个处理器中的每一个上（无需散布或收集）**确保：** 局部块 $\Re(C)$ $\Im(C)$1: $Q_1 \leftarrow$ 实数矩阵乘法(A_0、B_0) 3 个实数矩阵乘积中的第2: $Q_2 \leftarrow$ 实数矩阵乘法(A_1、B_1) 3 个实数矩阵乘积中的第3: $Q_3 \leftarrow$实数矩阵乘法(A_0+A_1、B_0+B_1) 对两个求和后的操作数执行第4: $\Re(C) \leftarrow Q_1 - Q_2$ ▷ 通过 3M 恒等式 (1.2) 组合5: $\Im(C) \leftarrow Q_3 - Q_1 - Q_2$

散布步骤后的组 g。 ∎

3.2 V2：原地 3M。算法 3.2 执行一次实数算法通信调度的单遍操作，移动复数值数据（字节数是实数值数据的两倍）；同时重构本地原地计算，使其使用 3M 恒等式 (1.2)，而不是直接四乘法公式：每个处理器从其驻留的A_0、A_1、B_0块中本地构造 、Q_2、Q_3 ，并将它们组合为 $\Re(C) = Q_1 - Q_2$、$\Im(C) = Q_3 - Q_1 - Q_2$。

引理3.2（V2 正确）。算法 *3.2* 可精确计算 $C = A \times B$。

证明。与引理 3.1 的代数论证相同：V2 形成相同的三个乘积 Q_1、Q_2、Q_3，并以相同方式将它们组合起来。唯一的区别是，这三个乘积都由每个处理器利用其已驻留的数据在本地计算，而不是分布在三个互不相交的组中计算。组合步骤没有变化，因此根据对 $Q_3 - Q_1 - Q_2$ 的相同展开即可得出正确性。 ∎

3.3 V3：标准。算法 3.3 使用与 V2 完全相同的通信调度（数据移动的内容、时间和数量均不变），但改为通过直接四乘法公式组合局部块：每个处理器在单次遍历中，利用其驻留块逐项形成 $\Re(C) = A_0B_0 - A_1B_1$和 $\Im(C) = A_0B_1 + A_1B_0$ （§1）。

引理3.3（V3 是正确的）。算法 *3.3* 精确计算 $C = A \times B$。

证明。由复数乘法的定义可直接得出：展开 $(A_0 + iA_1)(B_0 + iB_1)$ 后，直接得到 $\Re(C) = A_0B_0 - A_1B_1$ 和 $\Im(C) = A_0B_1 + A_1B_0$，无需验证中间恒等式。 ∎

4 分析 现在，我们利用第 2 节中的算术运算和

版权 © 20XX SIAM 未经授权的复制本文是被禁止的

算法 3.3V3：标准

要求： 局部块 A_0，A_1，B_0，B_1（实部、虚部部分 A，B），在每个处理器上均相同驻留 P 个处理器上（布局与 V2、算法 3.2 相同；无需散布或收集）

确保： 局部块 $\Re(C)$ $\Im(C)$

$\Re(C)$ $\Im(C) \leftarrow 0$ ▷ 初始化局部输出块

对于每个局部输出条目 (i, j) **执行**

  对于每个缩并索引 ℓ**执行** ▷ 累加按照 ℓ：实数算法自身的调度（§1），一个矩阵乘法，而非 3 次或 4 次独立运算

    a，$b \leftarrow (A_0)_{i\ell}$，$(A_1)_{i\ell}$ ▷ 实部、虚部 $A_{i\ell}$的

    c，$d \leftarrow (B_0)_{\ell j}$，$(B_1)_{\ell j}$ ▷ 实部、虚部 $B_{\ell j}$的

    $\Re(C)_{ij} \leftarrow \Re(C)_{ij} + ac - bd$ ▷ 平凡复数乘法公式，4 次实数乘法

    $\Im(C)_{ij} \leftarrow \Im(C)_{ij} + ad + bc$

  结束循环

结束循环

通信主导系数，给出 V1、V2、V3（第 3 节）的闭式成本表达式。对于算术指数为 ω 的实矩阵乘法基元，将 $F(n, P) = n^\omega/P$记为每个处理器的乘法次数（第 2.1 节）。将一个 实数乘积的算术运算主导系数记为 c_1（对于经典算法，根据第 2.2 节记为 $c_1 = 2$；对于 Strassen，根据第 2.5 节记为 $c_1 = 1 + q/3$）。将通过标准单遍调度应用直接 4M（或 k^2 个乘积）公式时的相应系数记为 c_2，将 3M/Toom-Cook-k 类似公式的相应系数记为 c_1^{TC}。

$$T_{V1}(n,P) = c_1\gamma F(n,P/3) + \beta BW(n,P/3) + \alpha S(n,P/3) + T_{\mathrm{redist}}, \tag{4.1}$$

其中，c_1 表示 the combined multiply-add 系数，即一个实数乘积的系数（$c_1 = 2$经典算法，§2.2）。$T_{\mathrm{redist}} = 2\beta\frac{n^2}{P/3} + 2\alpha$ 近似表示将 A_0、A_1、B_0、B_1 散布到各自组中的一次性开销。它还涵盖了在最后收集并合并 Q_1、Q_2、Q_3 的开销，这通过三个组中对应处理器之间的直接点对点交换实现。

$$T_{V2}(n,P) = c_1^{3\mathrm{M}}\gamma F(n,P) + 2\beta BW(n,P) + \alpha S(n,P), \tag{4.2}$$

其中，$c_1^{3\mathrm{M}}$ 是3M分解计算的局部算术运算（合并乘+加）系数。对于经典矩阵乘法，$c_1^{3\mathrm{M}} = 3 + 3 = 6$（§2.3）。对于 Strassen，$c_1^{3\mathrm{M}}$是§5中推导出的融合叶节点系数，它并不简单等于 $3c_1$。

$$T_{V3}(n,P) = c_2\gamma F(n,P) + 2\beta BW(n,P) + \alpha S(n,P). \tag{4.3}$$

对于经典矩阵乘法，$c_2 = 4$（§2.3）。$F(n, P) = n^3/P$同时承担乘法和加法主导系数（§2.2），因此（4.3）中的总算术运算系数为 $c_2^M + c_2^A = 4 + 4 = 8$。在提及合并的（乘法+加法）主导系数时，我们始终将其写作 $c_2 = 8$，这与§2.3 一致。

命题4.1（V2和V3共享带宽与延迟）。$T_{V2}(n, P) - T_{V3}(n, P) = (c_1^{3M} - c_2)\gamma F(n, P)$，这是纯粹的算术差异，不包含带宽或延迟项。

证明。由（4.2）和（4.3）可立即得出：两者都使用完全相同的 $2\beta BW(n, P) + \alpha S(n, P)$通信项，因为从4M到3M的局部重构只改变了对已驻留数据所执行的算术运算，而不改变通信内容或通信时机。 □

通信和算术运算并不是人们预期 V2 与 V3 存在差异的唯一方面：V2 的恒等式 (1.2) 引入了第三个乘积，$Q_3 = (A_0+A_1)(B_0+B_1)$，其操作数并不属于原来的四个输入块。一个朴素的实现会在连续三次调用现有实矩阵乘法例程之前，将两个和都具体化为全新的临时存储；这样在峰值时需要 8 个大小为 n^2/P 的块（4 个输入块、2 个输出块和 2 个求和临时存储，它们在计算 Q_3 的瞬间都同时驻留），比 V3 的 6 个增加了 33%。不过，这种朴素的排序并非必需。

命题 4.2（V2 的内存占用不必超过 V3）。*V2 存在一种实现方式，其峰值局部内存占用与 V3 相同：6 个大小为 n^2/P 的块（四个输入块 A_0、A_1、B_0、B_1 和两个输出块 $\Re(C)$ $\Im(C)$），只调用现有的实矩阵乘法例程，不使用额外的临时存储。*

证明。 以下顺序仅使用六个缓冲区 A_0、A_1、B_0、B_1、$\Re(C)$ $\Im(C)$}计算 $C = A \times B$，每当某个缓冲区的原始值不再需要时便重复使用它，且不使用其他缓冲区：

1. $\Re(C) \leftarrow$ 实数矩阵乘法(A_0, B_0)，得到 Q_1，并直接写入输出缓冲区。2. $\Im(C) \leftarrow$ 实数矩阵乘法(A_1, B_1)，得到 Q_2，并直接写入输出缓冲区。3. $A_0 \leftarrow A_0 + A_1$ 原地执行：A_0的原始值已不再需要，因为第 1 步已经使用了它。4. $B_0 \leftarrow B_0 + B_1$ 原地执行：B_0的原始值已不再需要，因为第 1 步已经使用了它。5. $A_1 \leftarrow$ 实数矩阵乘法(A_0, B_0)，利用现已被覆盖的求和缓冲区得到 $Q_3 = (A_0+A_1)(B_0+B_1)$，并将其写入现已空闲的 A_1 缓冲区；不再需要 B_0，B_1。

版权 © 20XX SIAM 未经授权的复制本文受到禁止

6. $\Re(C) \leftarrow \Re(C) - \Im(C)$，得到 $Q_1 - Q_2$，即正确的$\Re(C)$。

7. $\Im(C) \leftarrow A_1 - \Re(C) - 2\Im(C)$：由于 $A_1 = Q_3$，并且此时，$\Re(C) = Q_1 - Q_2$（来自步骤 6），而$\Im(C)$ 仍然成立 Q_2（未因步骤 6 而改变），因此计算得到正确的 $Q_3 - (Q_1 - Q_2) - 2Q_2 = Q_3 - Q_1 - Q_2$，即正确的 $\Im(C)$。

整个过程中完全不会使用第七个缓冲区，并且三次实矩阵乘法调用（步骤 1、2、5）均是对未修改的现有例程进行的普通调用。 ∎

因此，朴素八块排序是一种实现选择，而不是 3M 恒等式本身或重用现有实矩阵乘法例程所必然导致的结果：上述重新排序恰好实现了 V3 的内存占用，除了调用代码中的普通缓冲区管理外无需任何额外操作。

这清晰地分解了三方比较：V2 与 V3 仅比较常数 $c_1^{3\mathrm{M}}$和 c_2（§4.1），而 V1 与 V2 则完全取决于通信在处理器拆分下如何扩展（§4.2）。根据定理4.3，二者具有相同的计算时间。

定理4.3（浮点运算时间不变性）。T_{V1} 和 T_{V2} 当且仅当（经典矩阵乘法满足此条件；*Strassen* 的情形并不成立，见 §）：$3c_1^{3M} = 5c_1$ 的计算时间项相同，

$$3c_1\gamma F(n,P) = c_1\gamma F(n,P/3).$$

Proof. $F(n,P/3) = n^\omega/(P/3) = 3F(n,P)$, so $c_1\gamma F(n,P/3) = 3c_1\gamma F(n,P)$. ∎

对于经典矩阵乘法，定理 4.3 将 V1 与 V2 的比较归结为只比较通信。将 P 个处理器划分为三个并发组，每组$P/3$，其处理器总时间恰好等于在完整的 P 下，将经过 3M 重构的局部计算应用于整个问题所需的处理器总时间。

4.1 V2 与 V3：3M 到底值不值得做？ 对于经典矩阵乘法，$c_1^{3\mathrm{M}} = 6 < c_2 = 8$（§2.3），因此根据命题 4.1，$T_{V2} < T_{V3}$ 无条件成立。两者之间的差值是一个固定的绝对量，$2\gamma F(n,\ P)$；随着 $P \to \infty$，它在总时间中的占比趋于零，因为相同的通信项最终占据主导地位，但差值绝不会反转。这与 BLIS 的实际设计选择 [57] 相符。重构局部内核不会增加通信开销，因此没有理由不这样做。

对于 Strassen，这一结论反转了（§5）：融合叶系数 $c_1^{3\mathrm{M}}$超过 c_2，因此 $T_{V2} > T_{V3}$无条件成立；对于任意 P，原位 3M 都严格劣于标准方法。

4.2 V1 与 V2：拆分是否值得？

定理4.4（带宽交叉）。仅比较带宽项时，*V1* 的带宽项小于 *V2*，当且仅当带宽系数比 $3^{\theta(\omega)}$（*V1*，来自$BW(n,\ P/3) = 3^{\theta(\omega)}BW(n,\ P)$）小于 2（*V2*，来自复数数据加倍）：

$$3^{\theta(\omega)} < 2 \iff \theta(\omega) < \theta^* \equiv \log_3 2 \approx 0.6309.$$

证明。根据定理4.3（适用时），T_{V1} 和 T_{V2} 的计算项相等，因此单独比较带宽项 $\beta BW(n,\ P/3)$与 $2\beta BW(n,\ P)$，并对 $3^\theta = 2$ 求解 θ，即可得到所述阈值。 ∎

该阈值仅适用于带宽项，刻意不考虑延迟。它本身并不能决定最终胜者：带宽随P 减小，而延迟则增大，因此当 P 足够大时，无论 ω为何，两种布局中的延迟都会绝对占主导地位，真正的 $P \to \infty$极限由延迟而非带宽决定。

引理4.5（延迟主导）。假设 *V1* 将 P 个处理器拆分为 m个组（因此每组使用P/m个处理器进行计算），并且重新分配开销固定，其延迟贡献为 2α。当 $P \to \infty$时，对于带宽为 $BW(n,P) = \Theta(n^2/P^{\theta(\omega)}) \to 0$的任意布局：

1. 如果 $S(n,P) = \Theta(P^\rho)$ 属于某种 $\rho > 0$（多项式增长），那么在 T_{V1} 和 T_{V2}两种情况下，延迟最终都会超过带宽，从而得到 $T_{V1}/T_{V2} \to m^{-\rho} < 1$：*V1* 获胜，且优势幅度与 ω无关。*2.* 如果 $S(n,P)$（对数 P）（对数增长），延迟同样最终会超过带宽，但有限绝对差 $S(n,P) - S(n,P/m) = \log_2 m$ 此时会与重新分配开销的固定成本 2α竞争：$T_{V1}/T_{V2} \to 1$，当且仅当 $\log_2 m > 2$ 时 *V1* 领先。

证明。在情形 1 中，$BW(n,P) \to 0$，而 $S(n,P) \to \infty$，因此当 P足够大时，$\alpha S(n,P)$ 在 T_{V2} 中支配$2\beta BW(n,P)$，对 T_{V1} 也同样如此；延迟项之比为 $S(n,P/m)/S(n,P) = (P/m)^\rho/P^\rho = m^{-\rho}$，从而得到所述极限。在情形 2 中，同样的支配关系仍然成立，但 $S(n,P) - S(n,P/m) = \log_2 P - \log_2(P/m) = \log_2 m$ 恰好是一个常数；将 $T_{V1} - T_{V2} \to (\log_2 m - 2)\alpha$ 除以 $T_{V2} \to \infty$，得到 $T_{V1}/T_{V2} \to 1$：当 $\log_2 m > 2$ 时从下方逼近，否则从上方逼近。 ∎

推论4.6（真正的渐近优胜者）。对于二维为$P \to \infty$、对于三维为 $T_{V1}/T_{V2} \to \frac{1}{\sqrt{3}} \approx 0.5774$，且在两种情况下均独立于 ω。因此，对于每个算术指数，*V1* 在二维中始终是渐近优胜者，恰好领先 $1 - \frac{1}{\sqrt{3}} \approx 42.3\%$；对于每个 $\omega \leq 3$，*V2* 在三维中渐近表现至少同样出色，差距占总时间的比例逐渐趋于零，但从未反转。

证明。引理 4.5 对于 $m = 3$：在二维情况下，$S(n,\ P) = \Theta(\sqrt{P})$属于情形 1，且有 $\rho = \frac{1}{2}$，从而得到 $T_{V1}/T_{V2} \to 3^{-1/2}$；

版权 © 20XX，SIAM。未经授权的复制本文禁止

在三维情况下，$S(n,\ P) = \Theta(\log P)$ 属于情形 2，并且 $\log_2 3 \approx 1.585 < 2$，因此 V2 仍然领先。即使在 $\theta(\omega) > \theta^*$ 时也是如此，此时仅凭定理 4.4 的带宽比较会 favor V2：带宽只会推迟 V1 在二维情况下开始胜出的时间点，因为无论其自身符号如何，最终都会被延迟主导。□

经典二维在规模增大时更倾向于 V1，并收敛到推论 4.6 中普适的 $1 - 1/\sqrt{3} \approx 42.3\%$ 归约，与 ω 无关；经典三维则从不更倾向于 V1，而是收敛到有利于 V2 的消失间隙。定理 4.4 的带宽阈值在两种布局中仍然重要，但在二维中，它不影响最终胜出者的方向（这是无条件的），而是决定 P 必须足够大到什么程度，才能实现这一结果：与采用 $\theta(\omega) > \theta^*$ 的算法相比，采用 $\theta(\omega) < \theta^*$ (经典算法，Strassen) 的算法会在更小的 P 下达到二维渐近线，不过两者最终都收敛到相同的极限。

引理4.7（V1 优于 3M顺序）。一种在全部 P 个 $T_{seq}(n, P) = 3(c_1\gamma F(n, P) + \beta BW(n, P) + \alpha S(n, P)$ 处理器上顺序执行三个乘积的策略，在每个 $P \geq 3$下都被 *V1* 支配：$T_{V1} \leq T_{seq}$，因为计算项相同（定理*4.3*），而 $3^{\theta(\omega)} \leq 3$和上述延迟比均为 < 1。因此，我们不在主要比较中纳入 *3M*顺序。

4.3 完整的 2.5维范围：临界复制因子。定理 4.4 只比较两个端点，$c = 1$ 和 $c = P^{1/3}$。同样仅依据带宽的论证可推广到每个中间的 c，从而得到一个单一数值：对于固定且适中的 P，它将“拆分”与“不拆分”区分开来；与 §4.2 一样，真正的 $P \to \infty$ 极限是另一个问题，将在下文讨论。

推论4.8（临界复制因子，仅依据带宽）。令 $c = P^s$ 表示 $s \in [0, \frac{1}{3}]$，于是 $s = 0$为二维，$s = \frac{1}{3}$ 为三维。将 $M = \Theta(cn^2/P)$代入 *BDHS* 界 (2.1)，可得一个关于 s呈线性的带宽指数，

$$\theta(s;\omega) = \left(2 - \frac{\omega}{2}\right) + s\left(\frac{\omega}{2} - 1\right),$$

这在 $s = 0$ 处恢复了 $\theta_{2D}(\omega)$，并在 $s = \frac{1}{3}$ 处恢复了 $\theta_{3D}(\omega)$。仅比较带宽项时，当且仅当 $c < c^*$ 时，*V1* 优于 *V2*，其中

$$c^* = P^{s^*(\omega)}, \qquad s^*(\omega) = \frac{\theta^* - (2 - \frac{\omega}{2})}{\frac{\omega}{2} - 1}.$$

证明。使用上面的线性形式，求解 $\theta(s;\omega) = \theta^*$ for s。□

对于经典矩阵乘法（$\omega = 3$），这可简化为 $s^* = \log_3(4/3) \approx 0.262$，因此 $c^* = P^{\log_3(4/3)}$：仅比较带宽时，在从二维复制到完整三维复制的范围内，只要复制程度在以 $\log c$ 为尺度的大约 79% 以内，拆分（V1）就优于原地重构（V2）。

推论 4.9（真正的渐近优胜者，一般情形c）。当 $P \to \infty$ 在 $s \in [0, \frac{1}{3})$ 固定时，$T_{V1}/T_{V2} \to 3^{(3s-1)/2}$，与 ω 无关：对于每个 $s < \frac{1}{3}$，*V1* 都获胜，且随着 $s \to \frac{1}{3}$，优势连续缩小至0。只有在确切端点 $s = \frac{1}{3}$ 处，这一结论才会反转；根据推论 *4.6*，在该处 *V2* 仍以一个趋于零的差距在渐近意义上领先。因而，推论 *4.8* 的仅带宽阈值c^* 并不是真正的渐近交叉点；在 $[0, \frac{1}{3})$ 的内部不存在这样的交叉点，因为 *V1* 在整个区间内都获胜。相反，c^* 标示的是在有限 P 下带宽项自身符号发生变化的位置，因此仍是实践中相关的阈值，因为要观察到真正的渐近线，所需的 P 会随着 $s \to \frac{1}{3}$ 的增大而无限增长：例如，在 $s = 0.3$ 时，这要求 P，远远超出任何物理系统的范围，而 c^* 则能在 $P = 3^{18}$ 处正确预测有限-P 符号。

证明。对于 $s < \frac{1}{3}$，$S(n,\ P,\ c) = \Theta(\sqrt{P}/c^3) = \Theta(P^{(1-3s)/2})$：由引理 4.5 的情形 1，取 $m = 3, \rho = (1-3s)/2 > 0$，得到 $T_{V1}/T_{V2} \to 3^{-(1-3s)/2} = 3^{(3s-1)/2}$，严格小于 1。由于 $s \to \frac{1}{3}$，$\rho \to 0$，因此该项超过带宽并且 T_{redist} 无界增长的 P，恰在 $s = \frac{1}{3}$ 处发散；此时 $S = \Theta(\log P)$，推论 4.6 的情形 2 接替起作用。□

5 快速矩阵乘法的扩展将 (2.1) 应用于 $\omega = \omega_0 = \log_2 7 \approx 2.8074$，得到$\theta_{2D}(\omega_0) \approx 0.5963 < \theta^*$ 和 $\theta_{3D}(\omega_0) \approx 0.7309 > \theta^*$，与经典算法一样呈现相同的定性划分，但在不利的三维一侧留有更大 的余量（0.10，而经典算法为 0.036）。

推论 4.8 的临界复制因子让这一差距变得具体。将 $s^*(\omega)$ 取为 $\omega = \omega_0$，得到 $s^*(\omega_0) \approx 0.0857$，因此 $c^*_{\text{Strassen}} = P^{0.0857}$，而经典算法的值为 $P^{0.262}$：Strassen 的优势区间仅在从二维到完整三维复制的对数 c范围内约前进 26% 后便告结束，而经典算法则为 79%。这进一步揭示了有限内存情形：在任何具有哪怕适度复制的 2.5D 布局中，Strassen 的拆分策略远比经典矩阵乘法更早失去优势，因为 Strassen 较大的 $\theta_{3D}(\omega_0)$使得 s 中的余地小得多，因而更快就会越过 θ^*。正如推论 4.9 所示，这个更窄的区间是一个有限-P、仅带宽陈述：当 P 大到天文数字时，在整个 2.5维范围的内部，拆分对 Strassen 仍然有利，与经典算法一样。

算术运算常数需要更加仔细地处理。回想 §2.5，q 是每个 Strassen 递归层级中的矩阵加法次数，可分解为 $q = q_{\text{enc}} + q_{\text{dec}}$：编码从 A 和 B 的象限构造出 7 个操作数对，解码则将 7 个乘积重新组合到C的象限中。求解标准递归式可得

$$c_1 = 1 + \frac{q}{3}, \qquad c_2 = 6 + \frac{2q}{3},$$

对于一次实数 Strassen 乘积，或直接复数

版权 © 20XX 归 SIAM 所有。未经授权的复制本文被禁止

表 5.1：Strassen 算术系数，三种变体。

变体	q (编码/解码)	c_1 (V1)	c_2	$c_1^{3\mathrm{M}}$ (V2)	V2 与 V3
经典算法 [48]	18 (10/8)	7	18	19.67	比基准差 +9.3%
Winograd [59]	15 (8/7)	6	16	17.33	比基准差 +8.3%
Karstadt-Schwartz	**12（6/6）**	**5**	**14**	**15.00**	**+7.1% 更差**

$n \to \infty$。每个变体的 q （编码/解码）切分是一项既有结果，并按行注明引用来源。V1 的系数就是 c_1 其本身，因为它的三个组各只计算一个实数乘积；其与 V2/V3 的比较取决于 P 和布局，详见表 6.2。

Strassen，分别如此。后者将每层的 q 次加法都视为一次复数运算，成本增加一倍。直接复数 Strassen 是递归算法对 V3 的自然实现：它将 Strassen 自身的递归直接应用于复数类型数据，并采用朴素的复数叶节点。现在考虑 $c_1^{3\mathrm{M}}$，即原位 3M 的系数，或 融合叶节点 Strassen。这会在 Strassen 的递归拆分过程中传递两个实数通道，仅在标量基例处应用高斯技巧，并在每一层重新组合三个通道。编码阶段需要支付两次（每个通道一次），解码阶段需要支付三次（每个输出通道一次）。因此，每层的递归为 $c_1^{3\mathrm{M}}(n) = 7c_1^{3\mathrm{M}}(n/2) + [2q_{\mathrm{enc}} + 3q_{\mathrm{dec}}](n/2)^2$, , 叶节点值为 5 （高斯技巧的 3 次乘法 $+2$ 次加法。) 求解： g

$$c_1^{3\mathrm{M}} = 5 + \frac{2q_{\mathrm{enc}} + 3q_{\mathrm{dec}}}{3}.$$

表 5.1 为 §2.5 中介绍的三种 Strassen 变体——经典算法、Winograd 和 Karstadt-Schwartz——分别实例化了 c_1、c_2和 $c_1^{3\mathrm{M}}$。它采用经典切分 $q_{\mathrm{enc}} = 10$, $q_{\mathrm{dec}} = 8$ 以及 Karstadt-Schwartz 切分$q_{\mathrm{enc}} = 6$, $q_{\mathrm{dec}} = 6$。在这两种情况下， $c_1^{3\mathrm{M}} > c_2$：原位 3M 比标准更差。无论采用哪种变体的绝对加法次数，解码阶段的 $3\times$开销都会抵消乘法节省的成本。

根据文献验证的 $q = 12$ 拆分 [34], c_1 =5, $c_2 = 14$, $c_1^{3\mathrm{M}} = 15$：V2（原位 3M）无条件地比 V3（标准）差 7.1%，适用于任意 P和任意布局。这一排序与经典矩阵乘法相反，并且无论采用哪一种 Strassen 变体提供加法计数，都成立；使用正确的（而非通常假定的经典算法）常数，大致会使这一反转的幅度减半，但不会改变其方向。CAPS（§2.5）在 Strassen 的算术指数下实现了 V1 的分组拆分策略，从而得到通信项 $BW(n, P/3)$、 $S(n, P/3)$，其由上文推导出的相同 $\theta_{2D}(\omega_0)$、$\theta_{3D}(\omega_0)$决定。

6 数值结果 本节中的所有数值均使用 $\gamma = \beta = 2.5 \times 10^{-10}$s、 $\alpha = 10^{-6}$s、$n = 2000$。每张表都报告 §2.2 中的两种内存模式：二维（有限内存）和三维（无限内存）。负值表示成本降低 （左侧策略成本更低）；正值表示成本增加。全文分别报告 V2 与 V3 的比较（算术运算）以及 V1 与 V2 的比较（通信），将其视为两个独立比较，绝不合并为一个数值：其中一项降低并不意味着另一项也降低，两者也不能相加。以下所有数值均与作为补充材料提供的一致性检查模拟器进行了交叉核验：该模拟器根据每种布局的实际通信模式，在相同的 α-β-γ模型下，独立推导每个处理器的浮点运算次数、移动字节数、消息数量和峰值内存占用，并将这些结果与本文使用的闭式模型进行核对。这证实闭式表达式在代数上符合模型自身的假设；但这并不代表已根据真实的分布式实现对这些假设进行验证（§9）。

表 6.1：经典矩阵乘法：算术运算（V2:V3）、通信（V1:V2 带宽、延迟）。

布局	P	V2 [57]:V3	BW	延迟
2D	8.1×10^1	−25.0%	−13.4%	−42.3%
2D	5.9×10^4	−13.0%	−13.4%	−42.3%
2D	4.8×10^6	−0.04%	−13.4%	−42.3%
3D	8.1×10^1	−25.0%	+4.0%	−25.0%
3D	5.9×10^4	−23.5%	+4.0%	−10.0%
3D	4.8×10^6	−3.3%	+4.0%	−7.1%

$n = 2000$%。根据 §2.3、定理 4.4、引理 4.5 和推论 4.6。

6.1 经典矩阵乘法。表 6.1 印证了预测的分解。在小到中等规模的 P 下，对于两种布局，V2 相较 V3 的优势恰好为 25% 的降幅（$=(k-1)^2/k^2$ at$k = 2$，§2.3）；只有当通信占主导时，这一优势才逐渐趋近于 0%：三维情况下下降缓慢，二维情况下则很快。对于每个 P，二维的带宽固定为 −13.4%，三维固定为 +4.0%（定理 4.4）；二维的延迟固定为 −42.3%，而三维的延迟则随 P 增长而逐渐趋近于 0%（推论 4.6）。由于延迟在 $P \to \infty$ 时占主导，因此它决定了最终的优胜方案：在大规模二维场景中，在 V2 的基础上拆分为多个组；三维场景则绝不如此。

6.2 Strassen（Karstadt-Schwartz 切分）。表 6.2 展示了第 §5 节反转后的结果：V2 对 V3 在所有情况下均为正，因此贸然采用原位 3M 反而比什么都不做更糟；基于经典算法假设构建的 Strassen 实现需要修正，而不仅仅是扩展。对于

版权 © 20XX，SIAM。未经授权不得复制本文。

表 6.2：Strassen 算法：算术运算（V2:V3）、通信（V1:V2BW、延迟）。

布局	P	V2:V3	BW	延迟
2D	8.1×10^1	+7.1%	−3.7%	−42.3%
2D	5.9×10^4	+2.2%	−3.7%	−42.3%
2D	4.3×10^7	+0.00%	−3.7%	−42.3%
3D	8.1×10^1	+7.1%	+11.6%	−25.0%
3D	5.9×10^4	+6.2%	+11.6%	−10.0%
3D	4.3×10^7	+0.04%	+11.6%	−6.3%

$n = 2000$%。参见表 5.1 和推论 4.8。

二维而言，带宽在每个 P 下均为 +11.6%；对于三维则为 +11.6%，其幅度大于经典算法，因为 Strassen 的 $\theta(\omega_0)$ 在两种布局中都更远离 θ^*；然而，在两种布局中，延迟都与经典算法完全相同，且与 ω 无关（推论 4.6）：对于三维 *Strassen*，在我们评估的每个规模下，标准方案（*V3*）都是三种策略中最佳的。下文中的“Strassen”特指 Karstadt-Schwartz 备选基技术 [34], ，即表 5.1 中采用的、已证明加法最优的变体。

7 推广到多位数分割：Toom-Cook-k 和精度仿真

高斯恒等式（§2.3）和 Karatsuba 精度分割（下文）都是 Toom-Cook-k （§2.4）的 $k = 2$情形。我们将推导一般的 V2 与 V3、V1 与 V2 条件，均以 k 为函数，然后分别实例化。

7.1 一般系数与定理。将 c_1 记作一个实数乘积所需的合并（乘法+加法）算术运算，$c_2(k) = k^2c_1$ 和 $c_1^{\mathrm{TC}}(k) = (2k-1)c_1$ (§2.4)。

定理7.1（对于每个 $k \geq 2$，V2 都优于 V3）。对于任意 $k \geq 2$ 以及任何没有深层递归重组的基础算法，$c_1^{TC}(k) < c_2(k)$，其相对节省为 $(k^2 -(2k-1))/k^2 = (k-1)^2/k^2$。

证明。 $k^2 - (2k-1) = (k-1)^2 > 0$ 对于整数 $k \geq 2$。 □

这推广了在 $k = 2$(§2.3 以及下文的 Karatsuba 精度分割)已见到的 25% 节省；节省会随着 k 增大：当 $k = 3$ 时为 44%，当 $k = 4$时为 56%，当 $k = 5$时为 64%。这适用于任何不采用 Strassen 式递归结构的算法（不包括 Strassen，因为其自身的c_1 取决于其分割方式，见 §5）。

定理7.2（一般情形下的带宽交叉 k）。仅比较带宽项时，*V1* 的带宽项小于 *V2* 的，当且仅当$(2k-1)^{\theta(\omega)} < k$，即 $\theta(\omega) < \theta^*(k)$: = 对数$_{2k-1}$ k。

证明。 V2 移动 k 位数矩阵（带宽系数为 k，由复数据式通道加倍推广至 k 个通道）；V1 每组使用 $P/(2k-1)$ 个处理器（带宽系数为 $(2k-1)^{\theta(\omega)}$）。求解 $(2k-1)^\theta = k$ 关于 θ 的解。（V2 和 V3 共享带宽和延迟，见命题 4.1，因此这也相当于与标准 V3 进行比较。） □

$\theta^*(k) = \ln k/\ln(2k-1)$ 严格递增，$\theta^*(2) = \log_3 2 \approx 0.631$, $\theta^*(k) \to 1$ 当 $k \to \infty$（数值验证结果为：0.631，0.683，0.712，0.733，0.747，对应 $k = 2$, ……，6），因为高阶划分会将外部指数推向 $\omega \to 2$ 下取整值。如 §4.2 所述，该阈值仅决定带宽项：它本身并不能决定最终胜者，因为带宽会随P 减小，而延迟会增大，因此最终延迟将绝对占主导地位，不论 ω如何。

推论 7.3（真正的渐近优胜者，一般情形k）。对于二维，无论 ω 如何，只要 $P \to \infty$、~~$T_{V1}/T_{V2} \to \frac{1}{\sqrt{2k-1}}$，对每个 k（$1 - \frac{1}{\sqrt{2k-1}}$ 的归约)~~ 都成立。对于三维，$T_{V1}/T_{V2} \to 1$；消失间隙的符号由 $\log_2(2k-1)$ 与 2 的比较决定，同样与 ω 无关：只有在 $k = 2$（$\log_2 3 \approx 1.585 < 2$）时，*V2* 才保持渐近领先；对于每个 $k \geq 3$（$\log_2 5 \approx 2.322 > 2$），反之 *V1* 渐近领先，无论优势幅度多么微小。

证明。 将引理 4.5 中的 3 替换为 $m = 2k-1$：对于二维，$\rho = \frac{1}{2}$，从而得到 $T_{V1}/T_{V2} \to (2k-1)^{-1/2}$；对于三维，当且仅当 $\log_2(2k-1) > 2 \iff 2k-1 > 4 \iff k \geq 3$ 时，V1 领先。 □

推论7.4（高阶划分拯救三维）。对于每一个算术指数 ω，包括经典矩阵乘法和 *Karstadt-Schwartz Strassen*，$k = 2$ 对于三维而言在渐近意义下都不利，而$k = 3$ 依据推论 *7.3* 首先将其转为有利。这与 ω无关：决定因素是 $\log_2(2k-1)$ 与2的比较，而不是 $\theta^*(k)$ 与 $\theta_{3D}(\omega)$ 的比较。定理 *7.2* 中基于带宽的阈值仍然决定了 P 必须达到多大，这一渐近优胜者才会实现；在这一点上，它确实取决于 ω：对于经典算法，$\theta^*(3) = 0.683$ 已经超过 $\theta_{3D} = 2/3 \approx 0.667$，因此仅$k = 3$的带宽项就偏向 *V1*；对于 *Karstadt-Schwartz Strassen*，$\theta^*(3) = 0.683 < \theta_{3D}(\omega_0) \approx 0.731$，因此仅 $k = 3$的带宽项仍然偏向 *V2*，只有延迟在渐近意义下占主导时才会使 *V1* 领先。这意味着，*Strassen* 在 k 中的交叉点比经典算法的交叉点对应更大的P，尽管二者最终都会在相同的 $k = 3$处交叉。

将两个定理合并为一个有限-P 条件（V1 优于标准，当且仅当为正：

$$\mathrm{save}(n,P,k) = \frac{N(n,P,k)}{k^2+kx+y}, \quad \text{where}$$
$$N(n,P,k) = [k^2-(2k-1)] + [kx-(2k-1)^\theta x'] + [y-y'], \tag{7.1}$$

在$x = \beta_{BW}(P)/(n,P)$、$y = \alpha S(n,P)/(\gamma c_1 F(n,P))$，以及 x'，y'在 $P/(2k-1)$ 个处理器上具有相同的比率。在有限 P 下，符号由

版权 © 20XX SIAM 未经授权的复制本文受禁止

表 7.1：V1:V2 带宽/延迟、经典算法、三维布局，每 k 和 P。

k	$P=10^2$	$P=10^5$	$P=10^8$
2	$+4.0/-23.9$	$+4.0/-9.5$	$+4.0/-6.0$
3	$-2.5/-35.0$	$-2.5/-14.0$	$-2.5/-8.7$
4	$-8.5/-42.3$	$-8.5/-16.9$	$-8.5/-10.6$
5	$-13.5/-47.7$	$-13.5/-19.1$	$-13.5/-11.9$
6	$-17.6/-52.1$	$-17.6/-20.8$	$-17.6/-13.0$

$n=2000$。每个单元格：带宽/延迟变化，以百分比表示。负值= 归约，正值 = 增加。引自定理 7.2 和推论 7.3。

表 7.2：V1:V2 带宽/延迟、经典算法、二维布局，每 k 和 P。

k	$P=10^2$	$P=10^5$	$P=10^8$
2	$-13.4/-42.3$	$-13.4/-42.3$	$-13.4/-42.3$
3	$-25.5/-55.3$	$-25.5/-55.3$	$-25.5/-55.3$
4	$-33.9/-62.2$	$-33.9/-62.2$	$-33.9/-62.2$
5	$-40.0/-66.7$	$-40.0/-66.7$	$-40.0/-66.7$
6	$-44.7/-69.8$	$-44.7/-69.8$	$-44.7/-69.8$

$n=2000$。每个单元格：带宽/延迟变化，以百分比表示。负值= 归约，正值 = 增加。引自定理 7.2 和推论 7.3。与三维布局不同，此处两列在整个 P范围内均为常数，因为二维布局的带宽比和延迟比不依赖于 P（定理 7.2、引理 4.5）。

算术项（$k-1$）²（总是有利）和通信项（仅当$\theta(\omega) < \theta^*(k)$时才有利）共同决定；当 P渐近增大时，决定结果的是延迟而非带宽（推论 7.3）。

表 7.1 直接验证了推论 7.4，分别展示了带宽和延迟：从 $k=3$起，带宽为负值（有利），并且随着 k 增加而变得更加有利；而 $k=2$ 始终为正值（不利）；对于每个 k，延迟始终有利，但随着 P 增加而逐渐趋近于 0%。具体对于经典矩阵乘法，$\theta^*(k)$ 交叉 θ_{3D} 在 $k=3$时就已发生，因此仅带宽阈值与真正的渐近交叉在此处重合；对于二者不重合的算法（Strassen，第 5 节），k 中的渐近交叉仍发生在 $k=3$，但只有达到比仅带宽阈值所暗示的更大的 P 时才会发生。

7.2 Karatsuba 精度模拟实例。该框架无需修改即可应用于另一个无关场景：利用廉价的低精度（FP8/INT8）矩阵乘法，模拟高精度（FP64）矩阵乘积；在新型加速器上，每次运算的速度可提高 100 到 $500\times$ 倍 [52]。令 $A'=sA^{(1)}+A^{(2)}$,$B'=sB^{(1)}+B^{(2)}$ 表示以 s 为基数的两位拆分，Uchino 等人 [52] 将朴素的四乘积公式简化为三次乘法，这恰好是 Toom-Cook 的形式，其中将复数（1，i）基底替换为以 s 为基数的基底：$A'B'=s^2C^{(1)}+C^{(2)}+s\,(C^{(3)}-C^{(1)}-C^{(2)})$，适用于 $C^{(x)}:=A^{(x)}B^{(x)}$。定理 7.1 和 7.2 无需修改即可沿用：c_1=2, $c_1^{3\mathrm{M}}$=6, c_2=8；在相同的 $\theta^*=\log_3 2$ 下，二维场景中大规模时值得进行拆分，而三维场景中则几乎从不值得。

这里有两点需要注意：我们的评估使用了一个共享的 γ，但该恒等式的实际用途是访问更小的低精度 γ，因此真正的交叉 P^* 很可能比我们的数值所暗示的更小；此外，重组过程也存在 Higham [28] 在高斯情形中记录的同样的消去风险，而我们尚未针对这一实例进行独立检查，不过针对更广泛的 Ozaki-II 方案，近期已经出现了一般性的误差分析 [53]。

8 复合双线性算法的广度优先搜索放置 §4.2 中的 V1与V2问题及其 Toom-Cook-k 推广，都是一个更一般问题的实例：给定两个复合双线性算法，其中外层算法调用内层算法，应在哪一层分配唯一可用的处理器，使其进行拆分（用 CAPS 的 [36, 5] 术语来说，即执行一个广度优先搜索步骤）？当外层算法保持规模时（如 3M 和 Toom-Cook-k），答案是一个闭式阈值；而当外层算法缩减规模时（如 Strassen），则应采用不同的内存驱动判据。这正是 CAPS 自身的广度优先搜索/深度优先搜索选择尽管相似，却不属于我们的定理 7.2 的实例的原因。

令 $\mathcal{A}$ 通过 m_A 个辅助双线性乘积计算一个乘积（其元数），每个辅助乘积由原始操作数的 r_A 个通道 块的固定线性组合构成，并在 $O(N^2)$中重新组合：Toom-Cook-k 分解为 $r_A=k$ 个通道和 $m_A=2k-1$ 个乘积，高斯恒等式对应 $k=2$ 的情形（$r_A=2$，$m_A=3$）。令$\mathcal{B}$ 实现每个辅助乘积，其带宽指数为 θ_B，使用 P 个处理器。**广度优先搜索-$\mathcal{A}$**将 P 划分为 m_A组，每组包含 P/m_A，每组通过并发执行 $\mathcal{B}$ 来计算一个乘积（对于 Toom-Cook-k，恰为 V1）；**融合到-$\mathcal{B}$**则让所有 P 个处理器协同工作，按 $\mathcal{B}$的调度执行一次，并让所有 r_A 个通道一起移动（恰为 V2）。

定理8.1（一般广度优先搜索放置，保持规模 $\mathcal{A}$）。对于元数为 m_A、通道数为 r_A的保持规模 $\mathcal{A}$，与任意通信指数为 θ_B的 $\mathcal{B}$ 组合时，当且仅当 $\theta_B<\theta_A^*$ 时，广度优先搜索-$\mathcal{A}$ 的通信量小于融合到-$\mathcal{B}$：=对数$_{m_A}(r_A)$。

证明。与定理4.3一样，两种策略执行完全相同的总算术运算，因此比较仅涉及带宽。Fuse-into-$\mathcal{B}$ 在一次遍历中移动 r_A 个通道（系数 r_A）；BFS-$\mathcal{A}$ 每组使用 P/m_A 个处理器（系数 $m_A^{\theta_B}$）。求解 $m_A^{\theta}=r_A$ 得到 $\theta=\log_{m_A} r_A$。 □

这恢复了定理 7.2（$m_A=2k-1$，$r_A=k$），并且

版权 © 20XX SIAM 未经授权的复制本文文章，禁止进行

将定理 4.4（$k=2$）作为特殊情形；在 $m_A=3$，$r_A=2$ 处实例化后，恰好重现了表 6.1 中 V1 与 V2 的节省量对比。

定理 8.1 假设$\mathcal{A}$ 保持规模。Strassen 递归步骤并非如此：它将一个规模为N的乘积缩减为 $m_A=7$ 个规模为（$N/2$）的乘积，因此相关比较是广度优先搜索（7 个并发组，每组为 $P/7$）与深度优先搜索（在完整 P上依次处理这 7 个子问题）。类似计算表明，广度优先搜索的带宽是深度优先搜索的$m_A^{\theta_B-1}$ 倍，因此只要 $\theta_B<1$，广度优先搜索的通信量就不会更多；这里的每种算法都满足这一条件。这解释了 CAPS 自身在 [36, 5]采用广度优先搜索/深度优先搜索的选择：对于缩减规模的外层算法，广度优先搜索在通信方面几乎无条件地更有利，因此 CAPS 权衡的是内存，在预算允许 m_A倍内存时使用广度优先搜索，只有在内存受限时才使用深度优先搜索；这一标准与定理 8.1 的标准正交。

综合这两种情况：如果它保持规模且满足 $\theta_B<\theta_A^*$，就在 $\mathcal{A}$ 处使用广度优先搜索；否则融合 $\mathcal{A}$，并在更深一层提出相同的问题，即考察 $\mathcal{B}$ 自身。我们的外层算法（3M 或 Toom-Cook-k）始终保持规模，由一个内层算法（经典算法或 Strassen）构成，而后者不一定保持规模，因此这一递归一步即可终止：广度优先搜索步骤属于矩阵乘法之前，而不在矩阵乘法内部，即使内层算法是 Strassen 也如此，因为第 5 节的 θ_{2D}（ω_0） θ_{3D}（ω_0）正是定理 8.1 作为输入所需的θ_B。

9 讨论 我们将如何并行化 3M 复矩阵乘法技巧的问题拆分为两个独立的子问题：3M 是否值得采用（V2 与 V3，这纯粹是算术运算问题，因为两者采用相同的通信调度），以及是否值得重组通信以进一步利用 3M（V1 与 V2，这纯粹是通信问题，因为两者的计算时间相同）。对于经典矩阵乘法，局部采用 3M 总是值得的：其算术运算量减少 25%，与 BLIS 的设计相符；此外，对于大规模二维布局，将处理器拆分为多个并发组也值得采用，但三维布局则始终不值得。不过，二维与三维之间的差异，并非由最初看似决定这一点的、基于带宽的临界指数 $\theta^*=\log_3 2$ 决定，而是取决于带宽和延迟哪一项在 $P\to\infty$时占主导地位：在两种布局中，带宽都会随 P 缩小而降低，延迟则会增长，因此无论 ω如何，最终都会由延迟绝对主导。对于二维布局，V1 因此在任何算术指数下都具有普遍的 $1-1/\sqrt{3}\approx 42.3\%$ 延迟优势；而经典算法特有的带宽优势仅为 13.4%，它取决于 ω，并非普遍成立。这一优势并不只适用于满足 $\theta(\omega)<\theta^*$的情况；相反，θ^* 决定了 P必须达到多大，这一优势才会显现。对于三维布局，同样的延迟增长论证也适用，但 V1 重分布步骤的固定成本略微超过了其在延迟方面的固定优势，因此 V2 在渐近意义上仍至少同样优秀，对任意 $\omega\le 3$ 都成立；差距会随着 $P\to\infty$ 而消失，而不会反转。这一机制也可推广：对于 Toom-Cook-k （推论 7.3），二维结果完全推广成立（与 $1-1/\sqrt{2k-1}$、ω无关），而真正的三维交叉点从$k=2$ 移至 $k=3$，同样与ω无关；对于一般复制情形 c（推论 4.9），V1 在整个 2.5维范围的内部都是真正的渐近优胜者，而不只是到 c^*为止。不过，要观察到这一结果所需的 P 会随着 $c\to P^{1/3}$而无限增长，因此在任何有限规模下，c^* 仍是实际相关的阈值。

这项分析完全基于模型，我们尚未用真实的分布式实现对其进行验证。α-β-γ 模型本身忽略了与我们报告的百分比处于同一数量级的影响：V1 的三个并发组共享一个互连时产生的争用；负载不均衡；将 α，β，γ视为理想化常数与实际集合通信实现（散布、收集、广播-归约）之间的差距；以及全文中省略的 $O(n^2)$ 低阶项，这些项是否在我们的交叉规模下仍然是低阶项，并未得到单独确认。T_{redist} 尤其是一个粗略估计，并非基于具体的集合通信调度方案推导得出。在现有的二维/2.5维/三维库中，或直接在 CAPS 上实现一个基于 MPI 的版本，可以校准这些项，并通过测量验证交叉点 (7.1)；这留待未来工作。除此之外，我们的一致性检查模拟器（补充材料 §6）验证了这些闭式在代数上与对同一模型进行的独立离散核算相一致。这是模型准确性的必要条件，但并非充分条件，也无法解决上述遗漏。

本文提出的两类论断对 γ,β,α 的依赖各不相同。结构性 论断包括每一种渐近交叉的方向（在二维中，V1 对所有算术指数都优于 V2；在三维中则逊于 V2；$k=3$ 则使 Toom-Cook-k 在三维中扭转局面），这些论断是延迟项比率本身的 $P\to\infty$ 极限（引理 4.5）；而 γ、β,α 则完全抵消，在给出的比率（$1/\sqrt{3}$、$\log_2 3$ 与 2）中根本不会出现；因此，这些结论不受机器常数选择的影响。真正取决于 γ,β,α 以及带宽项的是交叉发生的位置：有限-P 阈值 c^*（推论 4.8），以及表 1.1 和 §6 中的精确百分比；这些百分比是在一种代表性设置下计算得出的，换用其他设置则会有所变化。应将每种交叉的渐近方向视为稳健的贡献，而将这些百分比视为某一工作点下的示例。

命题 4.2 的 6 块与 8 块内存比较是分布式内存核算，并不适用

版权 © 20XX，SIAM。未经授权不得复制本文。

于 BLIS 的单节点实现 [57]：BLIS 会将操作数打包到小型、按缓存分块的工作缓冲区中，而不是将每个操作数都作为一个持久块保存，因此其 3M 方法所需的任何额外工作空间都是临时的，且大小仅为缓存块级别，而非一个完整的额外块。

对于 Strassen 算法，若采用 Karstadt-Schwartz 加法次数，这一结论便会反转：原位 3M 严格劣于标准方案，因为其重组阶段无法像求值阶段那样在通道之间共享，因此在我们评估的所有尺度上，标准 4M 都是三维 Strassen 的最佳方案。这与两个假设有关：我们假设 Strassen 和基于 CAPS 的布局的延迟指数与经典算法相同；而 3M 已知的数值稳定性劣势 [28] 未纳入成本模型，这可能影响 V1/V2 在病态输入上的可行性；替代基 Strassen 的误差界 [46] 或许能让这一点更明确。其他快速算法（Pan、Schönhage 类型）是否也会呈现相同的 V2 与 V3 的逆转，也值得考察。

该框架可推广到高斯恒等式之外：将其应用于通过低精度仿真实现的 FP64 Karatsuba 高低位数字拆分时，同样的两个定理支配着相同的比较，并能正确预测出类似经典算法的排序，因为该恒等式没有深层递归重组。这里同样存在稳定性方面的注意事项，而且问题更为突出：据我们所知，这种重组的稳定性尚未经过单独检验。

推广至 Toom-Cook-k 后，可得到一个闭式优势条件，将算术运算节省（$k-1$）$^2/k^2$ 与临界指数 $\theta^*(k) = \log_{2k-1} k$结合起来，二者均随 k 增大：当 $k = 2$ 使三维处于不利一侧时，$k = 3$ 则使其渐近地对每一个 算术指数都变得有利；不过对于 Strassen 而言，由于 θ^* (3) 仍低于 $\theta_{3D}(\omega_0)$，这一优势只有在更大的 P 下才会显现，而不是像仅带宽阈值所暗示的那样（§4.2）。Nissim 等人提出的另一研究方向是，利用其替代基和 Toom-Graph 技术 [40], ，优化始终被压低处理的 $O(n^2)$ 低阶项，而不是将其视为可忽略项。

另一个独立的开放方向是容错：如何在 V1、V2 或 V3 上构建容错复矩阵乘法？容错矩阵乘法已经确立 [31, 14, 16, 11, 10], ，容错递归 Toom-Cook 也同样如此 [42, 41], ，但两者都没有涉及这种组合。改造其中任一种方法都意味着要确定容错层位于外层 3M 拆分、内层算法，还是二者之上，以及 V1 中已经彼此不相交的组能否兼作所需的冗余结构。

致谢。 本项目获得了欧洲研究委员会（ERC）在欧盟地平线2020研究与创新计划下的资助（资助协议编号：818252、101113120、101138056）。本研究获得了以色列科学基金会第 1354/23 号资助；该基金会由以色列科学院和人文科学学院创立。本研究还获得了科学加速器以及创新、科学与技术部科学前沿倡议的支持。

参考文献

[1]R. C. Agarwal、S. M. Balle、F. G. Gus- tavson、M. Joshi 和 P. Palkar，《一种并行矩阵乘法的三维方法》，IBM J. Res. Dev.，39（1995），第575–582页。

[2] A. Aggarwal、A. K. Chandra 和 M. Snir，并行随机存取机的通信复杂度，《理论计算机科学》，71（1990），第 3–28 页。

[3] F. Arute、K. Arya、R. Babbush 等，使用可编程超导处理器实现量子霸权，《自然》，574（2019），第 505–510 页。

[4]G. Ballard、J. Demmel 和 I. Dumitriu，通信最优的并行与串行特征值及奇异值算法，arXiv:1011.3077，（2011）。

[5]G. Ballard、J. Demmel、O. Holtz、B. Lipshitz 和 O. Schwartz，*Strassen* 矩阵乘法的通信最优并行算法，载于第 25 届 ACM 并行算法与体系结构研讨会（SPAA）论文集，2013 年。

[6]G. Ballard、J. Demmel、O. Holtz 和 O. Schwartz，最小化数值线性代数中的通信，SIAMJ.Matrix Anal.Appl.，32（2011），第 866–901 页。

[7]G. Ballard、J. Demmel、O. Holtz 和 O. Schwartz，快速矩阵乘法的图扩展与通信成本，J.ACM，59（2012）。文章 32。

[8] G. Beniamini 和 O. Schwartz，通过稀疏分解实现更快的矩阵乘法，载于第 31 届 ACM 算法与体系结构并行性研讨会（SPAA）论文集，2019 年。

[9] G. Bilardi 和 L. De Stefani，*Toom-Cook* 整数乘法的输入/输出复杂度，载于第 30 届 ACM-SIAM 离散算法研讨会（SODA）论文集，2019 年，第 2034–2052 页。

版权 © 20XX，SIAM 未经授权的复制本文被禁止

[10]N. Birnbaum、R. Nissim 和 O. Schwartz，面向快速矩阵乘法的高性能容错，载于 SIAM组合科学计算研讨会（CSC）论文集，2020 年，第 106–117 页。

[11] N. Birnbaum 和 O. Schwartz，容错、资源高效的矩阵乘法，载于《第八届SIAM组合科学计算研讨会（CSC）论文集》，2018年，第23–34页。

[12]L. S. Blackford, J. Choi, A. Cleary, E. D' Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker 和 R. C. Whaley，*ScaLAPACK*用户指南，SIAM，费城，宾夕法尼亚州，1997年。[13] M. Bodrato 和 A. Zanoni，关于 *Toom-Cook* 矩阵的最优性，技术报告，“Vito Volterra”中心，罗马第二大学，2006年。[14]G. Bosilca、R. Delmas、J. Dongarra 和 J. Langou，应用于高性能计算的基于算法的容错，《并行与分布式计算杂志》，69（2009年），第410–416页。[15] L. E. Cannon，用于实现卡尔曼滤波算法的细胞计算机，博士论文，蒙大拿州立大学，1969年。[16]Z. Chen 和 J. Dongarra，针对失效停止故障的基于算法的容错，《IEEE并行与分布式系统汇刊》，19（2008年），第1628–1641页。[17] J. Choi、J. J. Dongarra、R. Pozo 和 D. W. Walker，*ScaLAPACK*：面向分布式内存并发计算机的可扩展线性代数库，载于《第四届大规模并行计算前沿研讨会论文集》1992年。[18] S. A. Cook，函数的最小计算时间，博士论文，哈佛大学，1966年。[19] J. W. Cooley 和 J. W. Tukey，一种用于机器计算复傅里叶级数的算法，《计算数学》，19（1965年），第297–301页。[20] D. E. Culler、R. M. Karp、D. A. Patterson、A. Sahay、K. E. Schauser、E. Santos、R. Subramonian 和 T. von Eicken，*LogP*：迈向并行计算的现实模型，载于第四届 ACM SIGPLAN并行编程原理与实践研讨会（PPoPP）论文集，1993年，第1–12页。

[21]L. De Stefani，简讯：关于顺序和并行混合整数乘法算法的输入/输出复杂度，载于第34届 ACM 并行算法与体系结构研讨会（SPAA）论文集，2022年。

[22] E. Dekel、D. Nassimi 和 S. Sahni，并行矩阵与图算法，《SIAM计算杂志》，10（1981），第657–675页。

[23]J. Demmel、L. Grigori、M. Hoemmen 和 J. Langou，通信最优的并行与顺序 *QR* 和 *LU* 分解，《SIAM科学计算杂志》，34（2012），第A206–A239页。

[24] J. J. Dongarra、J. D. Croz、I. S. Duff 和 S. Hammarling，一组第*3*级基本线性代数子程序，《ACM软件数学杂志》，16（1990），第1–17页。

[25] A. Grama、A. Gupta、G. Karypis 和 V. Kumar，《并行计算导论》，Addison-Wesley，波士顿，第 2 版，2003 年。[26]L. Grigori、J. W. Demmel 和 H. Xiang，*CALU*：通信最优*LU*分解算法，SIAM《矩阵分析与应用杂志》，32 (2011)，第 1317–1350 页。[27] R. F. Harrington，《矩量法场计算》，Macmillan，纽约，1968 年。[28] N. J. Higham 等，《采用三次实数矩阵乘法计算复矩阵的方法的稳定性》，SIAM《矩阵分析与应用杂志》，13 (1992)，第 681–687 页。[29]M. Hoemmen，《通信规避 *Krylov* 子空间方法》，博士论文，加州大学伯克利分校，2010 年。[30]J.-W. Hong 和 H. T. Kung，《输入/输出复杂度：红蓝棋子游戏》，载于第 13 届 ACM 计算理论研讨会（STOC）论文集，1981 年，第 326–333 页。[31] K.-H. Huang 和 J. A. Abraham，《面向矩阵运算的基于算法的容错》，IEEE《计算机汇刊》，C-33 (1984)，第 518–528 页。[32] D. Irony、S. Toledo 和 A. Tiskin，《分布式内存矩阵乘法的通信下界》，《并行与分布式计算杂志》，64 (2004)，第 1017–1026 页。[33]A. Karatsuba 和 Y. Ofman，《自动机上的多位数乘法》，《苏联物理学文献》，7 (1963)，第 595–596 页。

版权 © 20XX SIAM 未经授权的复制本文章是被禁止的

[34] E. Karstadt 和 O. Schwartz，《矩阵乘法：快一点点》，《ACM 杂志》，67 (2020)。文章 1。

[35]T. Kouya，使用任意精度浮点算术加速复数矩阵乘法，arXiv:2307.06072，（2023）。

[36]B. Lipshitz、G. Ballard、J. Demmel 和 O. Schwartz，通信规避并行 *Strassen* 算法：实现与性能，载于 SC12 会议录，2012 年。[37] I. L. Markov 和 Y. Shi，通过收缩张量网络模拟量子计算，SIAM 计算杂志，38（2008年），第 963–981 页。[38] W. F. McColl 和 A. Tiskin，*BSP* 模型中的内存高效矩阵乘法，《算法》杂志，24（1999年），第 287–297 页。[39] D. Mukunoki、T. Ogita 和 K. Ozaki，面向众核架构的 *Ozaki* 方案：准确且可复现的 *BLAS* 例程，载于《并行处理与应用数学》（PPAM）会议录，2020 年。[40] R. Nissim、O. Schwartz 和 Y. Spiizer，将 *Toom-Cook* 算法中的输入/输出降至最低，载于《Euro-Par 2024：并行处理》，《计算机科学讲义》第 14803 卷，Springer，2024 年。[41] R. Nissim、O. Schwartz 和 Y. Spiizer，将容错 *Toom-Cook* 算法所需的处理器数量降至最低，载于第 37 届 ACM算法与体系结构并行性研讨会（SPAA）会议录，2025 年，第 514–524 页。[42]R. Nissim、O. Schwartz、Y. Spiizer、K. Agrawal 和 E. Petrank，容错并行整数乘法，载于第 36 届 ACM算法与体系结构并行性研讨会（SPAA）会议录，2024 年，第 207–218 页。[43]R. Nissim 和 Y. Spiizer，将 *Toom-Cook* 内积算法中的通信成本降至最低，载于《Euro-Par 2026：并行处理》，《计算机科学讲义》第 16782 卷，Springer，2027 年。[44]英伟达公司，*cuBLAS* 库。NVIDIA CUDA 工具包文档。网址：https://docs.nvidia.com/cuda/cublas/。[45] K. Ozaki、T. Ogita、S. Oishi 和 S. 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（2012年），第 95–118 页。

[46]*O. Schwartz*、*S. Toledo*、*N. Vaknin* 和 *G. Wiernik*，替代基矩阵乘法快速且稳定，载于 *Proc. IPDPS，2024*年，第*38–51*页。[47] *E. Solomonik* 和 *J. Demmel*，通信最优的并行*2.5*维矩阵乘法与*LU* 分解算法，载于 *Euro-Par 2011*，《计算机科学讲义》第*6853*卷，*Springer*，*2011*年，第*90–109*页。[48] *V. Strassen*，高斯消元并非最优，《数值数学》，*13*（*1969*年），第*354–356*页。[49] *A. Szabo* 和 *N. S. Ostlund*，《现代量子化学：高级电子结构理论导论》，麦格劳-希尔，纽约，*1989*年。[50]*A. Taflove* 和 *S. C. Hagness*，《计算电动力学：时域有限差分法》，*Artech House*，诺伍德，马萨诸塞州，第*3*版，*2005*年。[51]*A. L. Toom*，实现整数乘法的函数元件方案的复杂度，《苏联数学文献》，*3*（*1963*年），第*714–716*页。[52]*Y. Uchino*、*K. Ozaki* 和 *T. Imamura*，通过采用 *FP8* 量化的 *Ozaki-II* 方案进行双精度矩阵乘法模拟，*arXiv:2603.10634*，（*2026*年）。[53] *Y. Uchino*、*K. Ozaki* 和 *T. Imamura*，使用 *Ozaki-II* 方案的矩阵乘法模拟的误差分析，*arXiv:2602.02549*，（*2026*年）。[54]*L. G. Valiant*，一种用于并行计算的桥接模型，*Commun. ACM*，*33*（*1990*年），第*103–111*页。[55] *M. Valiev*、*E. J. Bylaska*、*N. Govind*、*K. Kowalski*、*T. P. Straatsma*、*H. J. J. Van Dam*、*D. Wang*、*J. Nieplocha*、*E. Apra*、*T. L. Windus* 和 *W. A. de Jong*，*NWChem*：面向大规模分子模拟的全面且可扩展的开源解决方案，*Comput. Phys. Commun.*，*181*（*2010*年），第*1477–1489*页。[56] *R. A. van de Geijn* 和 *J. Watts*，*SUMMA*：可扩展通用矩阵乘法算法，《并发：实践与经验》，*9*（*1997*年），第*255–274*页。[57]*F. G. Van Zee*、*T. Smith*、*F. D. Igual*、*M. Smelyanskiy*、*X. Zhang*、*M. Kistler*、*V. Austel*、*J. Gunnels*、*T. M. Low*、*B. Marker*、*L. Killough* 和 *R. A. van de*

版权 © 20XX，归 SIAM 所有。未经授权的复制本文章禁止。

GEIJN，通过 *3m* 和 *4m* 方法实现高性能复数矩阵乘法，《ACM数学软件汇刊》，44（2017年）。文章7。

[58]F. G. Van Zee 和 R. A. van de Geijn，*BLIS*：一种可快速实例化基础线性代数子程序功能的框架，《ACM 数学软件汇刊》，41（2015）。文章 14。

[59] S. Winograd，关于 2×2 矩阵的乘法，《线性代数及其应用》，4（1971），第 381–388 页。

版权 © 20XX，SIAM。未经授权的复制本文受到禁止