arXiv:2611.32440v1 [cs.DC] 24 Nov 2026

焦耳即浮点运算：在 GPU功耗墙处超越延迟隐藏，优化张量核心内核

Taowen Liu[*1]、Zhiwen Mo[*1]、Hongxiang Fan[1], AaronZhao[1*]共同贡
献 [1]伦敦帝国理工学院{tony.liu20, zhiwen.mo25, hongxiang.fan, a.zhao}
@imperial.ac.uk

摘要

对于平均板卡功耗接近 GPU 功耗上限的张量核心内核，其运行时间约等于每次调用消耗的能量除以该上限。即使计算隐藏了数据移动的延迟，数据移动仍会消耗能量；减少此类流量可以提高持续时钟频率并缩短执行时间，即使周期数不变。我们引入Joule，这是一个在张量内核设计与调度中考虑能量-频率耦合的框架。其成本模型结合硬件事件计数、物理数据通路能耗以及跨域的非线性电压缩放，在功耗上限下求解持续频率和运行时间。将该模型与跨存储层级和晶粒的数据移动下界相结合，可得到功耗感知理论峰值（PSoL），用于量化结构化差距和实现差距。该分析指导了在 B200 上使用完整张量内存（TMEM）容量的宽块内核，并通过细粒度异步旋转，将累加器读出与张量核心计算重叠。它还指导功耗感知空间与时间分块调度，在读共享、局部性和负载均衡之间进行权衡。给定测量的事件计数后，该模型在 B200 上实现了 0.69% 的平均能耗预测误差。在 60 种稠密层形状上，我们的通用矩阵乘法（GEMM）相较于通过穷举选择的 cuBLAS 算法，在 B200、H100 NVL 和 RTX PRO 6000 上分别取得了 4.94%、6.64% 和 2.12% 的几何平均加速比。对于运行在八块 B200 GPU 上的融合 Mega-MoE，所得优化相较于采用裸片本地专家权重的上游 DeepGEMM 基线，将性能提升了 2.60%–5.17%。我们将在论文发表后开源 Joule。

关键词

GPU内核、功耗墙、张量核心、功耗感知内核优化、动态电压频率调节

1 引言

张量核心内核主导着大语言模型训练和预填充工作负载 [24, 32, 61]。内核优化将内存访问与计算重叠，以保持张量核心处于忙碌状态，在大型通用矩阵乘法中达到约 95% 的利用率。这些内核还经常达到 GPU 的板卡功耗限制，或称为 功耗墙 [69, 70, 96]。随后，动态电压频率调节会降低时钟频率，使其保持在

功耗上限 P_{cap} [36]。功耗上限已达到 GPU 的最大值，因此若要进一步提速，就必须设计能更高效利用这一功耗预算的内核。

在功耗墙下，运行时间遵循 $t \approx E/P_{\text{cap}}$，其中 E 是内核的能量。即使计算完全隐藏了数据移动的延迟，数据移动仍会消耗能量：在 B200 上 PyTorch 默认的 cuBLAS 内核中，延迟隐藏的数据移动占总能量的 17.9%（图2）。减少这类流量可以降低能量消耗并提高

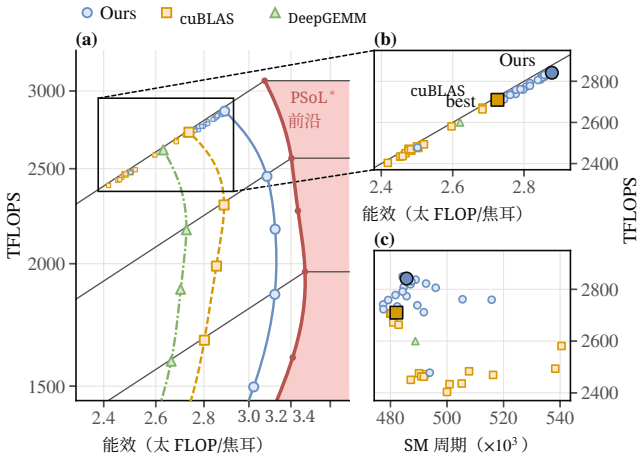


图1： 8192^3 **B200 上的 MXFP8 GEMM。（a）采用我们的 PSoL* 前沿（第3.2节）。我们观察到，（b）中的吞吐量与能效相关，而（c）中的 SM 周期数本身并不能反映这种相关性。**

持续时钟频率 [50]，，因此在固定功耗上限下，吞吐量取决于能效（图1b），而与流式多处理器周期数的相关性较弱（图1c）。在 8192^3 MXFP8 [26]通用矩阵乘法中，我们的内核执行的周期数比 cuBLAS 最佳实现多 0.6% [63]（图1c），但降低了数据移动能耗（图2），使每次调用的能量消耗减少 5.6%，运行速度提高 5.19%（图1b）。

这一观察揭示了延迟隐藏之外、考虑功耗墙的优化空间。即使计算完全隐藏了数据访问延迟，分块大小和多播仍会影响数据移动能耗。调度和实现也会影响冗余流量、同步和负载不均衡造成的能量与周期损耗。在 P_{cap}之下，这些成本会影响持续时钟频率，因此周期数相近的配置也可能有不同的运行时间（图1b）。新一代硬件（例如 Blackwell GPU）上的现有内核无法充分利用片上硬件。现有调优器 [4, 84, 87, 97] 会搜索内核实现所开放的配置。即使进行穷举搜索，其范围仍受限于该实现能够表达的调度方案。

要利用这一空间，需要预测某种机制如何同时改变能量和频率。性能模型估算周期数 [43, 55]，，而 GPU 功耗模型则估算给定时钟频率下的功耗 [47, 56, 86]。功率受限内核的时钟频率取决于其数据路径及其非线性的电压相关能耗。开发者还需要一个速度限制，以区分由内核结构造成的损失与可避免的调度开销。经典的 I/O 界 [39, 46, 76] 量化了最低数据流量，但要将

这些界转换为功率上限速度，就需要考虑每条物理数据路径的能耗成本（第 2.2.2节）。

内核还必须支持制定节能调度。更宽的瓦片可减少输入流量，但会留下更少的累加器空间用于缓冲，从而可能暴露流水线停顿。在流式多处理器之间共享读取操作可以节省能耗，但会增加同步开销并限制瓦片对齐。不相等的芯粒速度会进一步将局部性与负载均衡耦合起来，因为提前完成的流式多处理器仍会持续消耗空闲功耗。选择调度方案需要综合评估这些相互竞争的影响（第 2.2节）。

我们提出Joule，这是一个面向功耗墙的张量内核设计与调度框架。Joule 通过以下四项贡献，将能耗分析连接到扩展后的设计空间：

• **能耗—频率成本模型**。我们沿物理数据路径对硬件事件和每字节能耗成本进行建模，考虑跨域的非线性电压缩放，并求解由 P_{cap} 所允许的频率和运行时间。校准结合了微基准测试与实际内核测量。使用测得的事件计数后，该模型在 B200 上实现了 0.69% 的平均能耗误差（第 3.1 节和 5.2节）。

• **功耗感知速度上限与诊断**。我们推导了考虑双芯粒流量的多级 GEMM I/O 下界，并将其与成本模型相结合，得到功耗感知理论峰值 （PSoL），从而形成图 * 中的 PSoL1a 前沿。所得下界将结构化损失与实现损失区分开来，以指导内核重设计（第 3.2 节和第 4.1.1节）。

• **宽分块异步内核**。 在 Blackwell GPU 上，我们引入了一种新的内核结构，将全部 TMEM 专用于单个宽瓦片。细粒度异步轮换在无需双缓冲的情况下，使累加器读出与计算重叠，从而减少输入流量，同时在稠密 GEMM 和融合 Mega-MoE 中维持张量核心利用率（第 4.1.2节和4.2节）。

• **功耗感知空间与时间分块调度**。 我们的瓦片调度器利用整数规划，通过联合设置可变瓦片宽度、芯粒分配、读取共享和缓存驻留顺序，在多级能耗与周期成本之间进行权衡（第 4.1.3 节和 4.2节）。

在五个 Qwen3 模型的 60 种稠密层形状中， [92], Joule 的通用矩阵乘法性能优于通过逐形状穷举搜索选出的 cuBLAS 13.1 最佳算法。几何平均加速比在 B200、H100 NVL 和 RTX PRO 6000 上分别为 4.94%、6.64% 和 2.12%（第 5.3）。在八块 B200 GPU 上，采用相同设计原则的融合 Mega-MoE，相较于使用裸片本地专家权重的上游 DeepGEMM 内核，速度提升了 2.60%–5.17%（第 5.4节）。这些收益表明，扩展设计空间能够超越现有实现中的穷举调优。

我们计划将 Joule 开源，以帮助研究社区分析功耗墙瓶颈、开发能效感知的内核调度方案，并在这一设计空间上开展进一步研究。

2 背景与动机

2.1 背景

2.1.1 张量核心内核执行、调度与自动调优。张量核心内核（例如稠密 GEMM、MoE 分组 GEMM 和注意力机制）占据了大语言模型训练和预填充中绝大部分的执行时间与能量，因此即使是幅度不大的加速也极具价值

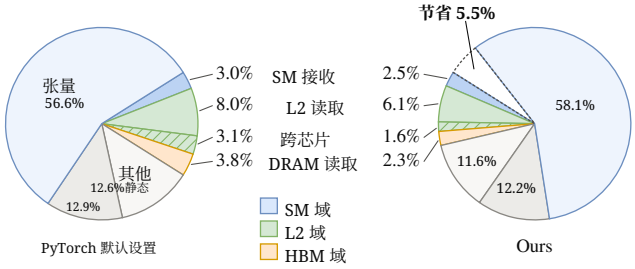


图 2：1000W 功耗上限下，B200 上一次 8192^3 MXFP8 GEMM 的能耗分解。随着数据移动占比从 17.9% 降至 12.6%，我们的内核相比 PyTorch 默认实现将总能耗降低了 5.5%。

。这些内核以分块的方式计算输出矩阵 [34,85, 88]：输入分块经由高带宽内存和二级缓存暂存到片上内存 [11], , 同时 MMA 指令在束寄存器（H100 NVL）或专用张量内存（B200 上的 TMEM）中累加部分和 [16, 57]„ 通常会预留两个累加器缓冲区（乒乓缓冲） [65, 71]，以便在计算的同时进行尾处理读取。为最大限度地复用数据并隐藏延迟，内核会结合 (1)片上共享，通过 GPC 请求合并器 [66, 68] 或 SM 集群多播，消除不同流式多处理器间重复的二级缓存读取；以及 (2) 分块调度，通过安排分块的执行顺序（二级缓存交错布局），使共享操作数留在二级缓存中，并将异步内存提取流水线安排在 MMA 周期之后。由于这些分块、共享和调度选择会随矩阵形状相互影响，cuBLAS、CUTLASS、ThunderKittens 和 DeepGEMM 等最先进的库 [28, 63, 65, 78] 依赖启发式方法或黑盒自动调优 [18, 31, 62, 75, 84, 87, 89]——在硬件上对候选内核进行性能分析，为每种形状选择最快的配置——即使对于大型 GEMM，也已能让张量核心在 ~95% 的周期数内保持活跃。

2.1.2 多芯粒内存 *GPU*。 现代数据中心 GPU 越来越多地采用多芯粒封装 [5, 30]。例如，NVIDIA B200 集成了两个通过高带宽跨芯粒链路连接的计算芯粒；每个芯粒都拥有自己的流式多处理器、63MiB 的 L2 分片和本地高带宽内存。因此，传送至某个流式多处理器的一个字节可能来自本地 L2（单独提供，或通过 GPC 合并/集群多播进行合并）、经由跨芯粒链路从远端 L2 获取，或来自高带宽内存 [30, 51]。默认情况下，物理页会交错分布在两个芯粒上，而跨芯粒访问会在本地 L2 [59] 中复制缓存行（使共享数据的有效 L2 容量减半）；不过，也可以通过 CUDA 局部性域 [64]将张量显式放置在特定芯粒上。

2.1.3 功耗受限内核与动态电压频率调节。张量核心内核经常触及板卡功耗限制 P_{cap} （B200 上为 ~1000W，H100 NVL 上为 ~400 W）。随后，固件会通过动态电压频率调节（DVFS）降低受调节域（如流式多处理器和二级缓存）的时钟频率 f_d和供电电压 v_d（f_d），直至功耗达到 P_{cap}。因此，内核可持续运行的频率取决于其每周期能耗，而运行时间则由总能量决定：$t \approx E/P_{\text{cap}}$。

2.2 动机：功耗墙下的差距与挑战

2.2.1 延迟隐藏流量的隐性能耗成本。传统的内核优化主要依据执行周期数进行分析：只要内存访问被张量核心计算隐藏、不造成流水线停顿，关键路径之外的操作就被视为没有成本。然而，在功耗墙下，延迟隐藏操作仍会消耗能量，并直接降低时钟频率（图 1 和 2）。即使在 PyTorch 默认的 cuBLAS 内核中，B200 上一个 8192^3 通用矩阵乘法的能量中，17。9% 也消耗在已被计算隐藏的数据移动上：其中 3。0% 用于向流式多处理器传输字节数，8。0% 用于二级缓存读取，3。1% 用于跨芯片传输，3。8% 用于 DRAM 读取。回收这些隐性能量后，动态电压频率调节便可提高时钟频率、缩短完成时间，但由此带来的加速并非线性：随着较少的事件使频率 f_d 升高，供电电压 $v_d(f_d)$ 会沿非线性曲线急剧攀升，使该域中剩余事件的能耗增加，并部分抵消节省下来的能量。

2.2.2 现有工具的不足。现有的性能和功耗工具无法为功耗上限下的内核优化提供指导：

• **硬件计数器和性能分析器：**诸如 Nsight Compute [66] (ncu) 之类的工具会报告周期级指标和硬件事件计数，但无法将整板功耗归因于单个事件或物理数据路径。更糟的是，为了在多次重放时收集确定性计数器，ncu 默认会将 GPU 时钟锁定在固定的基准频率，从而抹去动态电压频率调节（DVFS）造成的降频，并错误评估那些以略多的周期数换取更低能耗和更高功耗限制下速度的设计。

• **性能分析模型：**性能模型和模拟器 [7, 40, 43, 48, 53, 55, 60, 82] 会估算给定时钟频率下的执行情况。Roofline [90] 等经典界限采用固定的计算上限，而在功耗受限的动态电压频率调节下，这一上限可能无法达到。功耗屋顶线 [49] 将性能与能效联系起来，但细粒度能耗归因仍是一个尚未解决的缺口。经典的输入/输出下界 [2, 8, 39, 46, 67, 76] 量化了最小流量，但要将其转换为功耗限制下的速度上限，就需要知道各路径的能耗。因此，开发者仍然需要一个速度上限，以区分由分块大小或共享配置造成的损失，以及由冗余流量和流水线停顿造成的损失。

• **GPU 功耗模型：** 基于事件的功耗模型 [6, 47, 56, 86]解决的是逆问题：它们预测给定频率下的功耗，而不是求解在 P功耗上限约束下允许的稳态频率和运行时间。此外，它们无法感知新的 GPU 微架构或多域非线性电压缩放，因此会产生 7%–36% 的误差，无法用于指导内核设计。

• **选择启发式方法与自动调优：** 库选择和经验自动调优 [63, 73, 87] 可以通过测量捕捉能量导致的速度差异，但仍受限于现有实现所提供的配置。即使进行穷举选择，也无法获得那些必须改变内核结构才能实现的性能提升。

2.2.3 功耗墙感知内核优化的挑战。 即使能耗核算准确且掌握了理论界限，在功耗墙下重新设计和调度内核仍面临三方面的矛盾：

• **能量与周期数之间的权衡：** 减少数据移动的硬件机制往往会引入额外的等待或同步周期。例如，使用更大的多播集群可以通过跨 SM 共享减少二级缓存读取字节数，但会迫使更多流式多处理器在同步屏障处相互等待，并加剧尾波不平衡。

• **优化层级之间的冲突：**在某个内存层级上节省能量的选择，往往会限制另一个层级上的优化。例如，最大化分块宽度以减少 SM 接收字节数，会限制分块的划分方式及其为实现二级缓存共享所采用的对齐方式。

• **跨单元速度不对称性：**传统调度会在计算单元之间均匀划分工作，假设它们的速度一致。然而，在功耗墙约束下，不同的晶粒（其启用的流式多处理器数量并不对称）和不同的 GPU 会维持不同的稳态频率。

3 Joule 框架

图 3 展示了 Joule 工作流，它通过硬件事件计数连接三个组件 $\vec{n}$。 能耗—频率成本模型 （第 3.1 节）针对每款 GPU 仅需校准一次，并将任意事件计数向量 $\vec{n}$ 转换为 P_{cap} 下的稳态时钟频率和运行时间。 功耗感知理论峰值（*PSoL*）和差距分解 （第 3.2 节）将$\vec{n}^{\min}(\sigma)$ 沿每条物理数据路径所需的最小事件计数输入成本模型，以求出 P_{cap} 下的速度上限，并与测量计数比较以定位瓶颈。 内核设计和瓦片调度 （第 3.3 节）通过修改内核消除已诊断出的瓶颈，并查询成本模型以解决相互冲突的调度选择。

3.1 功耗墙感知能耗–频率成本模型

Joule 成本模型基于两项核心设计。首先，它为每次数据传输所经过的特定物理数据路径分配能耗成本。其次，它根据每个时钟域测得的非线性电压曲线缩放事件能耗，并将工作频率视为 P_{cap} 下的未知量，从而直接根据事件计数求解运行时间。

3.1.1 能量建模与频率求解。

多域非线性能耗模型。 GPU 包含多个彼此独立的电压域，例如 SM、二级缓存和高带宽内存域，其中每个域内的单元共享供电电压和时钟频率。当内核达到 P_{cap} 时，动态电压频率调节会降低受调节域的频率和电压，从而减少这些域内的单事件能耗。背景能耗 $\bar{P}t$ 来自恒定的背景功率 $\bar{P}$，且仅取决于执行时间 t。动态能耗来自硬件事件，包括张量核心操作、CUDA 核心指令，以及沿各条物理数据路径传输的字节数。对于电压域 d 中的事件 u，其单事件能耗与相对电压的平方成正比

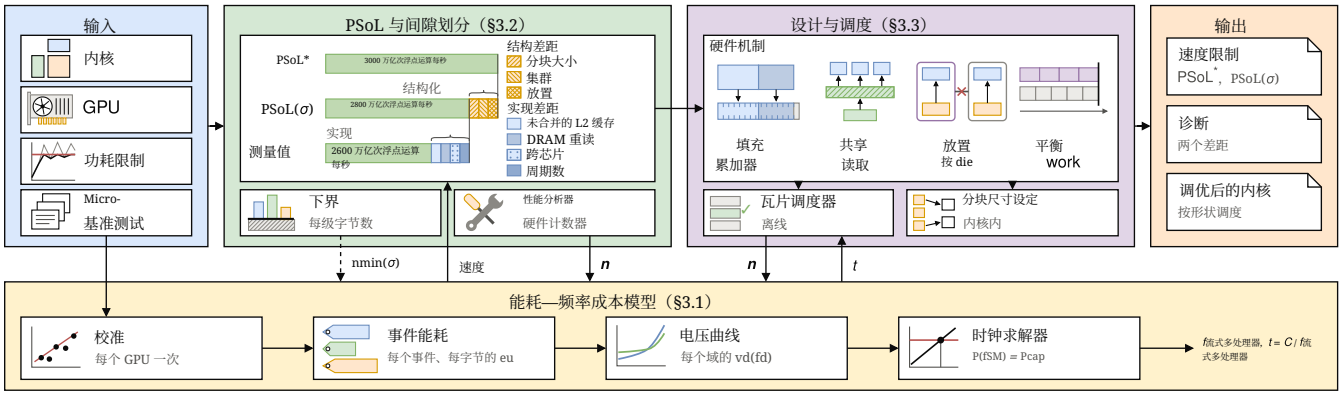
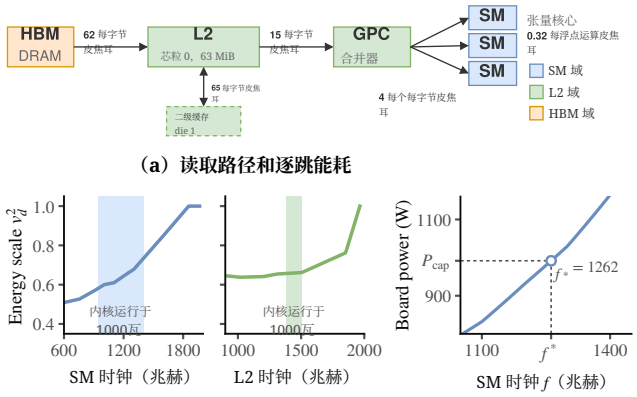


图 3：Joule 概览。能耗—频率成本模型利用硬件事件计数 $\vec{n}$ 来预测功率上限运行时间、推导 PSoL 速度限制，并指导内核设计和瓦片调度。



（a）读取路径和逐跳能耗

（b）流式多处理器和二级缓存域中的能耗缩放 （c）在 P_{cap} 下进行频率求解

图 4：B200 上的 Joule 成本模型：（a）物理路径，（b）电压域能耗缩放 v_d^2，其中内核在约 1000瓦时的频率以阴影标示；（c）在 1000瓦下为8192^3 通用矩阵乘法求解稳态时钟。

相对电压 [14] $v_d(f_d)^2$，归一化为 1 在最高频率下：

$$E = \bar{P}t + \sum_d v_d(f_d)^2 \sum_{u \in d} e_u n_u, \tag{1}$$

其中，f_d 是域 d 的工作频率，n_u 是事件 u 的计数，e_u 是其在最高频率下的基准能耗成本。在现代 GPU 上，电压曲线 $v_d(f_d)$ 在低时钟频率下较为平缓，在接近峰值频率时则会急剧变陡。在 P_{cap} 下，供电电压的下降幅度远小于时钟频率，因此，假设事件能耗恒定或采用线性电压–频率缩放都会导致较大的预测误差。

建模架构特定物理路径。我们使用微基准测试，识别会显著影响各 GPU 架构能量消耗的物理数据路径和电压域行为。对于双芯粒 B200（图4a），模型将 GPC 请求合并器处的 L2 至 SM 传输拆分为 L2 读取项和针对每个接收端流式多处理器的接收项，以捕捉共享读取所节省的能量。模型还按来源将本地 L2 未命中拆分为跨芯片 L2 传输和 HBM DRAM 读取，以捕捉双芯片数据放置的影响，并根据 P_{cap} 下的动态电压频率调节模型对能量进行缩放。我们将同样的原理应用于具有分离式 L2 半切片的 H100 NVL，以及 L2缓存与流式多处理器共用同一电压域的 RTX PRO 6000。

在 P_{cap}下进行频率求解。我们发现，二级缓存时钟 f_{L2}和固定的 DRAM 时钟 f_{DRAM} 都可以表示为 SM 时钟 f_{SM}的函数。一旦动态电压频率调节将各域的时钟频率关联起来，公式（1）中的能量就变成 SM 时钟 f_{SM}的单变量函数 $E(f_{\text{SM}})$。对于执行 C 个 SM 周期的内核，其运行时间为 $t(f_{\text{SM}}) = C/f_{\text{SM}}$，平均功率为$P(f_{\text{SM}}) = E(f_{\text{SM}})/(C/f_{\text{SM}})$。在 P_{cap}下，我们的求解器会找出平均功率等于功率上限时的稳态频率：

$$\frac{E(f_{\text{SM}})}{C/f_{\text{SM}}} = P_{\text{cap}}. \tag{2}$$

由于每秒周期数和供电电压都会随时钟频率升高，板卡功耗会随 f_{SM} 严格单调递增（图4c）。因此，方程（2）有唯一的根。我们通过布伦特方法 f_{SM}求解 [12]，直接根据事件计数得到运行时间 C/f_{SM} 和能量$E(f_{\text{SM}})$。该公式还量化了第2.2节所述的非线性抵消效应：减少硬件事件会降低 $E(f_{\text{SM}})$ 并提高允许的时钟 f_{SM}，但随之而来的 $v_d(f_d)^2$ 上升会部分抵消节省的能量。

3.1.2 模型校准。 校准事件能耗成本 e_u 和电压曲线 $v_d(f_d)$ 仅使用真实内核或仅使用微基准测试进行校准，结果都不准确。在真实内核中，硬件事件计数之间存在很强的共线性。例如，在多次调用中，L2 读取字节数和 SM 接收字节数会同步变化，因此普通最小二乘法无法分离它们各自的能耗成本 [38]。微基准测试每次只隔离一种事件，从而消除共线性，但它们会在未受限的时钟频率下单独驱动硬件单元；若直接将其结果用作模型系数，就会引入系统性偏差。为同时解决这两个问题，我们在真实内核上拟合最小二乘模型，并采用 Theil混合估计 [83]，将独立的微基准测试测量值作为加权先验纳入模型，以锚定共线项。由于公式（1）对

能耗成本 e_u 呈线性关系，而对电压曲线 $v_d(f_d)$ 呈非线性关系，因此我们使用变量投影（VarPro） [33]求解这一混合最小二乘问题。对于给定的电压曲线参数，VarPro 通过线性最小二乘 e_u 以闭式形式求解 [52, 79], , 从而只需将低维的电压曲线节点交由外层非线性优化器处理。最后，由于启用硬件性能计数器会改变功耗，我们会在禁用计数器的情况下测量稳态功耗和频率 [13, 93] ，并在单独的性能分析轮次中采集事件计数。

3.2 功耗感知理论峰值（PSoL）与差距分解

由于成本模型仅以事件计数作为输入，因此既可以评估现有内核，也可以计算任何内核在 P_{cap} 下所能达到的最高速度，以及测量所得的内核在哪些方面损失了速度。

多级内存与双芯粒流量下界。对于张量核心内核，算法确定了浮点运算次数总量和输入大小。在硬件容量限制和静态内核结构 σ（包括分块大小、多播集群宽度和芯粒本地操作数放置）给定的情况下，瓦片复用决定了各内存层级所需的最小事件计数 $\vec{n}^{\min}(\sigma)$ ：

SM 接收字节数： 使用能够放入片上累加器和缓冲区中的最大瓦片，可最大限度地提高输入复用率，并最大限度地减少 SM 接收字节数。**L2 读取：** 假设每个集群内进行完全多播，并在集群间实现理想的 GPC 请求合并，即可确定最小 L2 读取字节数。**DRAM 读取：** 经典的缓存感知 I/O 下界 [39, 76] 给出了冷输入的最小 HBM 读取量，以及操作数超出 L2 容量时必须进行的重读量。**跨芯粒读取：** 对于双芯粒 GPU（图4a），并行通信下界 [46]给出了芯粒本地放置下的最小跨芯片流量；而默认页面交错则根据工作在两个芯粒之间的划分方式，确定跨芯片流量所占的比例。

结构差距与实现差距。 对于静态内核结构 σ，将其最小事件计数 $\vec{n}^{\min}(\sigma)$ 输入成本模型（第 3.1 节）并求解式（2），在 P_{cap} 条件下可得条件 *PSoL*，记作 PSoL(σ)。对所有有效结构 σ 上的 PSoL(σ) 取最大值，即可得到 GPU 的 绝对 *PSoL*，记作 PSoL*:

$$\text{PSoL}^* = \max_{\sigma} \text{PSoL}(\sigma). \tag{3}$$

将内核的实测速度 S_{meas} 与两个上限进行比较，可将其相对 PSoL* 的差距分为两部分：

$$\text{PSoL}^* - S_{\text{meas}} = \underbrace{\left(\text{PSoL}^* - \text{PSoL}(\sigma)\right)}_{\text{Structural Gap}} + \underbrace{\left(\text{PSoL}(\sigma) - S_{\text{meas}}\right)}_{\text{Implementation Gap}}. \tag{4}$$

结构差距： PSoL* –PSoL(σ)。这种损失由所选的静态结构 σ决定，例如分块大小较小、集群宽度次优或采用交错芯粒放置；若不改变 σ，就无法弥补。**实现差距：** PSoL(σ) – S_{meas}。固定 σ后，这种损失源于 GPC 请求合并不完善、分块遍历不理想导致额外的 DRAM 读取或跨芯片重复读取，以及未能隐藏的流水线或屏障周期。集群宽度在内核启动时固定，属于结构差距；而 GPC 合并取决于哪些分块在同一 GPC 上并发执行，因此属于实现差距。

3.3 PSoL 引导的内核设计

逼近 PSoL* 需要同时弥合结构差距和实现差距。要弥合结构差距，内核必须使用宽瓦片来填满累加器，在多个流式多处理器之间共享 L2 读取，并将操作数放置在 die本地内存中。要弥合实现差距，还必须规划分块调度、die 和 GPC 分配以及执行顺序，以减少缓存重复读取，并确保所有流式多处理器几乎同时完成。由于这些选择可能相互冲突，例如更大的多播集群虽能减少 L2 读取，却会增加屏障等待，Joule 会使用其成本模型评估这些权衡，并在 P_{cap} 下选出运行时间最短的调度方案。第 4 节将这一方法应用于静态形状 GEMM 和动态路由的 Mega-MoE，第 5.5.1 节则量化这两种差距。

4 Joule 驱动的内核设计与调度

张量核心利用率较低的内核很少能达到 P_{cap}。因此，我们将Joule 应用于两类核心张量核心工作负载，它们持续撞上功耗墙：采用 MXFP8 的静态形状稠密矩阵乘法和 FP8 GEMM（第 4.1 节），以及采用 Mega-MoE 的动态路由融合专家混合（第 4.2 节）。

4.1 稠密 GEMM

我们以 B200 GEMM 作为主要案例研究。由此得到的内核机制和瓦片调度器同样适用于 H100 NVL 和 RTX PRO 6000 上的 GEMM，以及 B200 上的融合 Mega-MoE（第4.2节）。

4.1.1 结构差距诊断与条件 *PSoL*。我们首先诊断 B200 上 GEMM 的局限性（图 13，详见第 5.5.1 节）。一个 PSoL（σ）的条件上限 会随着 $256 \times W$ 瓦片宽度的增加而提高W，因为在流式多处理器上，宽瓦片每次浮点运算接收的字节数更少。在 cuBLAS 能够实现的最佳多播配置下，其 256×256结构每个瓦片仅使用一半 TMEM，因此在几何平均值中可达到 PSoL 的 94.1%*，峰值达到 86.3%；而 256×480 MXFP8 瓦片可达到 PSoL 的 96.2%–97.2%*。我们通过采用全 TMEM 宽瓦片和细粒度异步轮换，在不牺牲尾声阶段重叠的情况下弥合这一结构差距（第 4.1.2 节）；随后通过功耗感知瓦片调度，弥合由未合并的 L2 读取、DRAM 和跨芯片重复读取以及额外周期造成的剩余实现差距（第 4.1.3 节）。

4.1.2 内核机制：全 *TMEM* 宽瓦片与细粒度异步轮换。

双面板宽瓦片。 在 B200 上，一对流式多处理器协同处理一个 256 行的输出带，而`tcgen05` MMA 指令会将乘积累加到片上 512 列的张量内存（TMEM）中，如图 5左侧所示。[1] 完成瓦片的所有 K 列块后，尾声必须先从 TMEM 中读出结果，才能让下一个瓦片覆盖这些列。因而，传统方案会将 TMEM 分成两个 256 列的缓冲区， X 和 Y，并交替使用 256×256 两个缓冲区处理瓦片，使一个缓冲区读出数据时，另一个缓冲区进行累加（图 5a）。由于任一时刻只有一半 TMEM 用于累加，要计算图 5a 中的 1024 个输出列，需要四个瓦片，并将操作数 A 加载四次。受协同调度的启发，我们

[1]图 5展示了 BF16 的情况，它无需缩放因子，并使用全部 512 个 TMEM 列形成每个面板包含 256 列的 256×512 瓦片。我们的内核还支持 MXFP8，该格式会保留 512 个 TMEM 列中的 32 个用于块缩放因子，并将瓦片的最大尺寸限制为 256×480，其中面板 X最多包含 256 列，面板 Y最多包含 224 列。

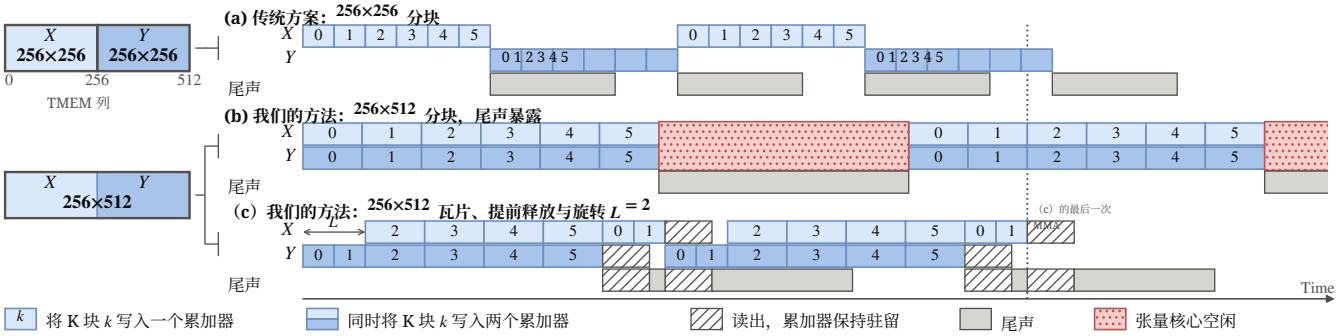


图 5：B200 上的全 TMEM 宽瓦片执行示意图。图中以 BF16 为例，所有 512 个 TMEM 列上分布着 256×512 个瓦片；而 MXFP8 会预留 32 列存放缩放因子，从而形成 256×480 个瓦片。(c) 中的提前释放 TMEM 和 L 块轮转消除了 (b) 中的读出停顿，使 MMA 时间与 (a) 中的乒乓缓冲相匹配，同时加载次数减少了 A 一半。

转而在单个宽瓦片中，同时使用 TMEM 的两半进行累加，分别作为左右累加器 X 和 Y。在图 5b 的每个步骤中，两个累加器都会重用同一份已提取的 A 数据，并分别与 B 中各自对应的列块相乘。每个宽瓦片每个 K 块完成的工作量是原来的两倍，因此两个宽瓦片就能在相同的总 MMA 时间内完成 1024 列的计算，同时只需加载 A 两次。然而，一个瓦片占用全部 TMEM 后，就没有多余的缓冲区来隐藏读出开销。图 5b 中，两个面板同时完成第 5 个块后，下一个瓦片必须等尾声从 TMEM 中读出数据后才能启动，导致张量核心停顿。

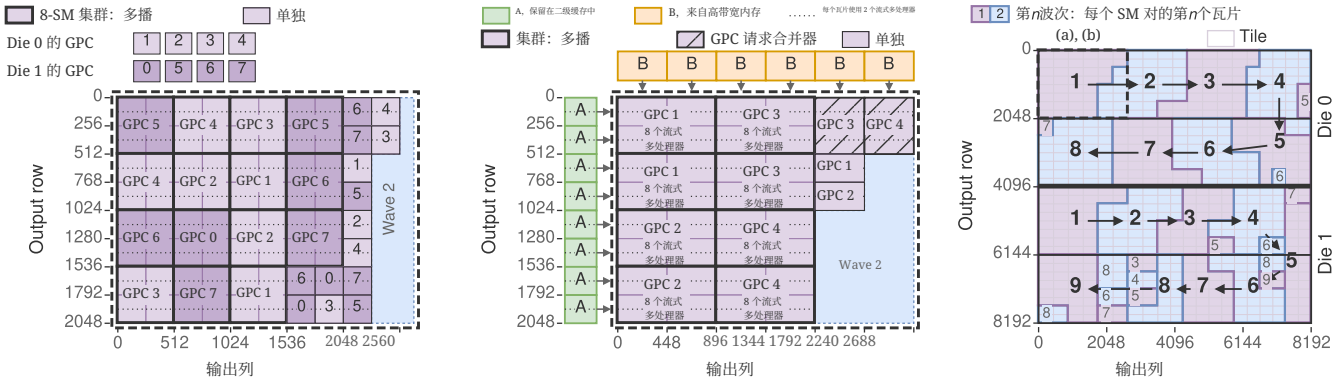
通过细粒度异步轮换隐藏尾声开销。我们通过两种机制消除这些停顿（图 5c）。首先，尾声将已完成的面板从 TMEM 复制到线程束寄存器，并立即释放其 TMEM 列；释放之后，较慢的 HBM 回写在灰色区域中异步进行。其次，我们将 X 的 K 个块重新排序，从 0、1、2、3、4、5 调整为 2、3、4、5、0、1；我们将此称为轮换（图 5c 中的 $L = 2$）。Y 提前开始执行 L 个块，并单独执行块 0 和 1。随后，两个累加器共同执行块 2、3、4、5，并共享一次对 A 的加载。最后，Y 提前完成 L 个块，并在 X 执行剩余块 0 和 1 的同时读出结果。这种轮换需要将 A 的前 L 个块加载两次，每个累加器各加载一次；其余所有 $K - L$ 个块则只加载一次并由两者共享。L 是该内核设计空间中的一个参数。

细粒度瓦片宽度。此外，每个 SM 对都会从内存中读取一份静态调度列表，其中指定了每个瓦片的输出坐标以及 X 和 Y 的确切宽度。支持细粒度瓦片宽度，可以避免在不可整除的矩阵维度上进行浪费的越界计算，并将每个瓦片的宽度、跨 die 划分、SM 到 GPC 的分配以及执行顺序暴露给我们的瓦片调度器（第 4.1.3 节）。

4.1.3 功耗感知瓦片调度器。我们的内核展现了一种新颖的功耗感知瓦片调度设计空间。传统方案不会根据晶粒和 GPC 来规划瓦片（图 6a），这会造成跨芯片流量，并使 CTA 集群之外的流式多处理器无法通过 GPC 请求合并器实现合并。它们还会导致较快的晶粒在内核执行结束时处于空闲状态（图 7，上图）。给定内核配置后，我们的调度器会通过分析求解瓦片调度分为四个步骤：首先，为避免跨芯片流量并匹配跨芯片速度，调度器根据测得的每芯片吞吐量，按比例将输出工作负载划分到两个晶粒上 [9]（图 6c），并将每个晶粒的输入分区固定在其本地内存中 [64]。其次，在确定每个晶粒的工作负载后，整数规划 [44]会将输出维度分解为可变分块宽度，以尽量减少越界浪费工作，同时为较快晶粒上的 SM 对分配更宽的分块，使两个晶粒能同时完成（图 7，下图）。第三，为最大限度地提高 GPC 内部读取共享（图 6b），我们的调度器同时利用多播和 GPC 请求合并器来合并 L2 读取，并对仍处于多播集群之外的流式多处理器进行空间打包。第四，为最大限度地提高 L2 重用（图 6c），调度器将连续瓦片分组，直至达到单芯片 L2 容量，并按蛇形波次顺序遍历这些瓦片，从而使共享操作数在多个波次间持续驻留于 L2 中。

4.1.4 内核选择。我们的内核配置会产生大量组合，因此我们使用启发式方法来筛选配置。为了无需运行 GPU 即可评估这些配置，CPU 模型会根据合成的瓦片调度预测每个候选项的硬件事件向量 $\vec{n}$，而 Joule 成本模型（第 3.1 节）会预测其功率上限运行时间。我们可以直接在 CPU 上选出排名最高的候选项，无需运行 GPU；也可以将候选项缩小到少数排名靠前的选项，再在设备上进行稳态验证。

4.1.5 跨架构适配：*H100 NVL* 和 *RTX PRO 6000*。在 H100 NVL 和 RTX PRO 6000 上，warp 寄存器累加器将瓦片限制为 256×128，因此我们的性能提升来自对功耗感知瓦片调度器和成本模型引导的选择（第 5.3 节）的适配。**H100 NVL：**我们的内核采用 256×128 个瓦片并进行双向多播。我们使用瓦片调度器，并利用 H100 NVL 成本模型选择集群模式、半切片 L2 组大小和遍历顺序。**RTX PRO 6000：**在这款不支持集群多播或跨 SM 合并的单芯片 GPU 上，我们采用瓦片调度器，并使用 RTX PRO 6000 成本模型，在 256×128 与更窄瓦片之间进行权衡，以实现尾波平衡，并选择可最大限度减少 DRAM 重读的 L2 交错顺序。



(a) 芯粒 0 和芯粒 1 上的传统空间映射　**(b) 我们在芯粒 0 上的空间映射**　**(c) 输出的时间顺序以及芯粒输出拆分**

图 6：B200 上 $8192 \times 8192 \times 16384$ 通用矩阵乘法的瓦片调度。硬件良率地板扫描会导致各 GPC 的流式多处理器数量不相等，且不总是 CTA 集群大小的整数倍；其中 GPC 1、2 各有 18 个流式多处理器，GPC 3、4 各有 20 个流式多处理器。与 (a) 的传统调度相比，(b) 我们的调度采用 256×448 宽瓦片，将 8-SM 集群限制在单个 die 和 GPC 内，并在同一 GPC 中配对剩余的流式多处理器，以实现请求合并。(c) 在不同 die 间拆分输出行，并展示波执行顺序。

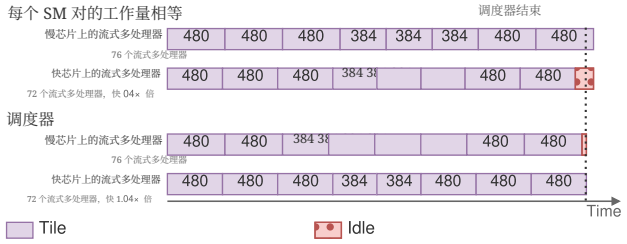


图 7：在 B200 的两个 die 间按速度比例分配工作。每个 die 的功耗预算相近，因此流式多处理器更多的 die，其每个流式多处理器的功耗预算更低，运行频率也就更低。

4.2 Mega-MoE

我们的功耗墙感知内核调优方法和宽分块机制不仅适用于独立通用矩阵乘法，也适用于其他受功耗限制的张量核心工作负载。我们以 DeepSeek-V4-Pro 的专家并行混合专家层为目标 [29]。通过分析基线内核的事件计数，Joule 显示，其性能未达 PSoL* 同样源于结构差距（有效分块过窄且权重读取未共享），以及实现差距（余数块填充和 TMEM 释放停顿）。我们引入四项运行时改进，同时严格保持批次不变输出：

宽有效分块与旋转。 按照我们的宽瓦片 GEMM 设计（第 4.1.2 节），每个流式多处理器针对一个已获取的 token 分块计算两个权重子块。一个 token 分块因此可服务于两倍数量的权重行，从而使每个输出对应的流式多处理器接收的 token 字节数减半，并降低 $\vec{n}^{\min}(\sigma)$ 以缩小结构差距。由于每个流式多处理器的两个子块需要两个累加器，我们会轮换这两个子块的累加器，以隐藏尾处理读取，同时避免扩大实现差距。

更深的权重流水线。 跨两个子块的一步会消耗两个权重阶段，因此，六阶段权重流水线不再能隐藏加载延迟，并会扩大实现差距。我们将权重流水线加深至八个阶段，并通过缩短 token 流水线、每次提取半个 token 分块，为其腾出共享内存。

内核内瓦片大小选择。 路由只有在运行时才能确定每个专家的词元数量，因此内核会在内核内选择瓦片大小：将每个专家的词元均匀划分为多个瓦片，每个瓦片的高度都是 16 的倍数，从而消除基线中带填充的余数瓦片及其浪费的 MMA 周期。消除这些填充瓦片可减少张量核心能量的浪费和额外周期，从而直接缩小动态路由下的实现差距。

权重共享。 4-SM集群中的两个 SM 对会认领同一专家、同一权重列中相邻的词元瓦片，因此，从二级缓存加载的一个权重瓦片会多播给这两个 SM 对。集群多播将 $\vec{n}^{\min}(\sigma)$ 中最低限度的二级缓存权重字节数减半，并进一步缩小与 PSoL* 的结构差距。

与上游内核一样，每个 die 都从 die本地内存处理每个权重块的一半。旋转会固定每个投影沿 K 的累加顺序，因此输出在不同批次间保持不变；它们与上游内核的区别仅在于累加顺序。我们将 token路由与分块调度的协同设计留待未来研究。

5 评估

我们从四个方面评估Joule 。首先，在第 5.2 节中，成本模型对未见过的内核的功耗受限能耗、运行时间和相对加速比的预测有多准确？其次，在第 Joule 节中，经其调优的稠密 GEMM 是否优于不同 GPU 架构上的最先进库？我们的内核选择与现有自动调优器和通用调优器相比如何？5.3？第三，在第 5.4 节中，相同的设计原则是否也适用于动态路由、多GPU 融合 Mega-MoE？

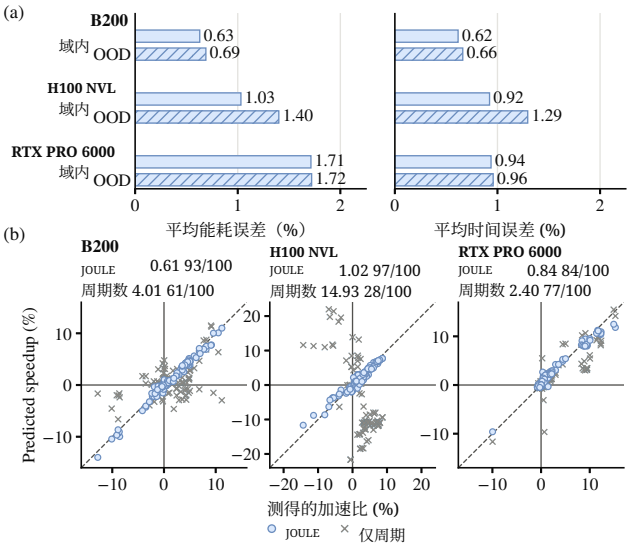


图 8：Joule 成本模型在三种 GPU 上的准确度。单内核的域内和域外误差保持在 1% 左右，成对排名优于周期比。

第四，在第 * 节中，各种机制分别弥补了多少 PSoL 差距，又各自贡献了多少？5.5?

5.1 评估设置

我们使用 CUDA 13.4 和 cuBLAS 13.1，在功耗为 1000瓦的 NVIDIA B200、400W 的 H100 NVL 以及 600瓦的 RTX PRO 6000 上评估 Joule。在 B200 和 RTX PRO 6000 上测试 MXFP8 GEMM，1×128 在 H100 NVL 上测试 [58] FP8 GEMM，并在 B200 和 H100 NVL 上测试 BF16 GEMM；测试覆盖五个 Qwen3 模型的全部 60 种稠密层形状，[3, 92] 范围为$M \in \{2K、4K、8K、16K\}$，另包括 DeepSeek-V4-Pro 的 [29] 融合 Mega-MoE（第 5.4 节）。我们报告相对于 PyTorch 默认 cuBLAS 启发式方法、通过逐形状穷举扫描获得的 cuBLAS 最佳结果以及 DeepGEMM 的几何平均加速比 [28]。我们使用服从正态分布的输入 [35, 37] 并清空二级缓存；每个数据点取三次 3 秒 CUDA Graph 运行的中位数，每次运行前冷却 45 ℃，丢弃前 1 秒，以测量动态电压频率调节的稳态表现（第 5.3.1 节评估初始升频窗口）。

5.2 能量–频率成本模型评估

本节使用测量得到的硬件事件计数来评估 Joule 成本模型 $\vec{n}$；而第 5.3.2 节 则评估该模型与解析事件计数估计结合后在内核选择中的表现。每个 GPU 只需进行一次校准，所有结果均基于留出数据报告。

能量与运行时间准确度。图8a 报告了在三个 GPU 上、两种数据划分下的能量与功率上限运行时间预测误差。域内划分采用真实内核测量点进行 10 折交叉验证，而域外（OOD）划分则将整个算法族排除在训练集之外，例如 DeepGEMM、宽瓦片 GEMM、融合 Mega-MoE 或 FlashAttention [95]。在两种划分和全部三个 GPU 上，能量与运行时间的平均误差均维持在 1% 左右。RTX PRO 6000 上的误差略高，这是因为该显卡会随温度变化在两条电压曲线之间切换，使短时微基准测试无法达到重负载内核所处的电压范围，

表 1：B200 上的模型比较与结构消融。底部区块中的每一行都在前一行的基础上增加一项特征。Joule的能耗误差是图 8a 中按留出系列计算的误差；其他行则各自留出一个内核。

	能耗误差（%）		成对加速比	
模型	Mean	Max	均方根（pp）	Hits
每 FLOP 固定能耗	5.89	25.14	–	–
每周期固定能耗	3.68	13.52	4.01	61/100
Wattchmen [86]	21.27	30.42	2.70	74/100
AccelWattch [47]	7.23	23.02	2.66	78/100
No-$v(f)^2$ 基线	2.32	10.49	2.25	78/100
+ V–f曲线	0.95	3.97	1.97	83/100
+ GPC 请求合并器	0.82	4.49	1.42	87/100
+ 局部性感知（Joule）	**0.69**	**2.39**	**0.61**	**93/100**

因此静态项和内存路径项主要依赖先验值。

成对加速比排名。图8b 比较了 Joule 成本模型与单独使用周期比的效果，针对每种 GPU 的 100 对留出候选对进行评估。在 B200 和 H100 NVL 上，Joule 识别较快内核的效果远优于周期比。在 RTX PRO 6000 上，Joule 将均方根误差从 2.40 个百分点降至 0.84 个百分点，并在差异超过 1 个百分点的 49 对中命中 48 对。

表 1 比较了 Joule 与 B200 上的启发式基线和现有功耗模型，并对第 3.1.1 节和 3.1.2 节中的组件进行消融实验。Wattchmen [86] 仅使用加速时钟下的微基准进行校准，并将功耗受限能耗高估了 21.27%。AccelWattch [47], 即使采用我们的校准流程重新拟合，仍假设电压–频率线性缩放，在 B200 和 H100 NVL 上分别产生 7.23% 和 36% 的误差。使用我们的 Theil 混合校准重新拟合 Wattchmen 的恒定能耗公式（No-$v(f)^2$ base）可将误差降至 2.32%，因为功耗受限内核的运行时钟频率范围较窄；而省略微基准先验会使最坏情况下的分布外误差从 2.70% 升至 3.57%。加入测量得到的非线性 $v_d(f_d)^2$ 曲线（+ V–f曲线）使单次调用误差降幅最大，降至 0.95%；而对 GPC 请求合并器和双芯粒局部性进行建模（+ GPC 请求合并器和 + 局部性感知）则带来了成对排名方面的提升，将均方根误差降至 0.61 个百分点，能耗误差降至 0.69%。

5.3 GEMM 评估

5.3.1 通用矩阵乘法结果。

主要结果。 图 9 和 10a 显示，我们的通用矩阵乘法在三款 GPU 上始终优于所有基线。与 cuBLAS 最佳性能、PyTorch 和 DeepGEMM 相比，我们的内核在 **B200** 上的几何平均加速比分别为 4.94%（60 种形状中胜出 59 种）、6.26% 和 8.64%。在 **H100 NVL** 上，加速比分别为 6.64%（60 种形状全部胜出）、17.40% 和 5.12%。在 **RTX PRO 6000** 上，相较于 cuBLAS 最佳性能和 PyTorch，加速比分别为 2.12%（60 种形状中胜出 59 种）和 11.44%。性能提升在大尺寸 K 形状上最为显著，而少数性能回退仅出现在 $M = 2048$的情况下，此时每个 SM 对处理的瓦片约为一个，无法摊销启动开销。

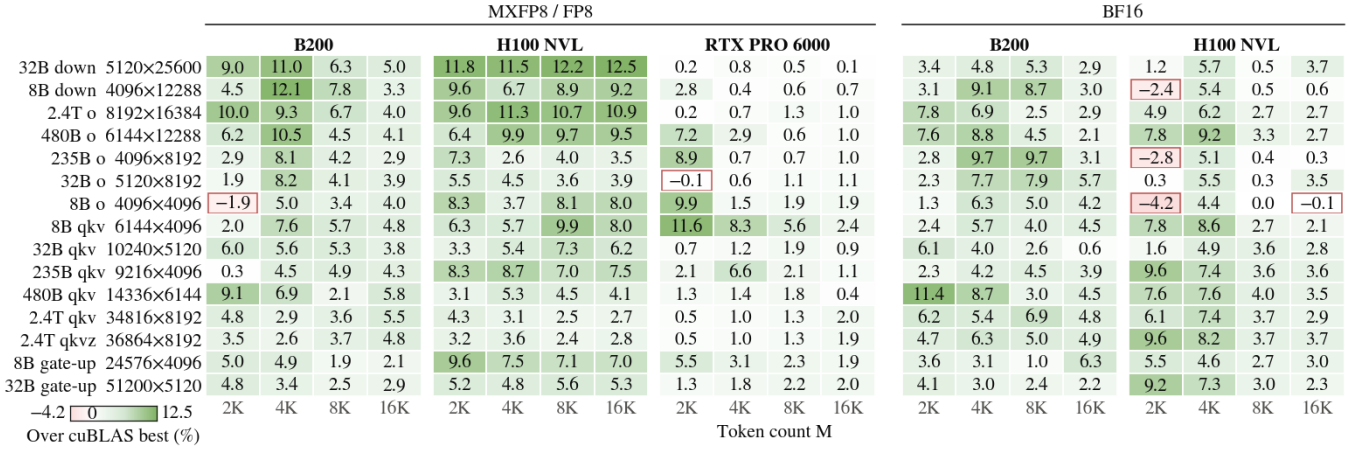
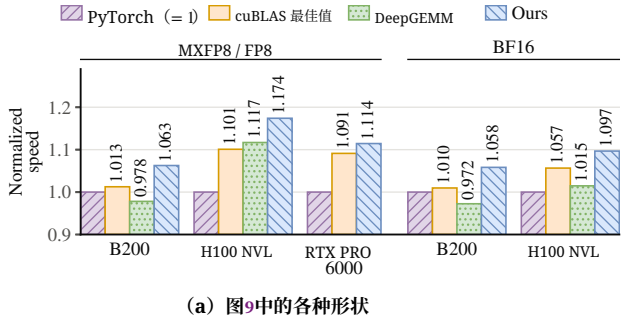
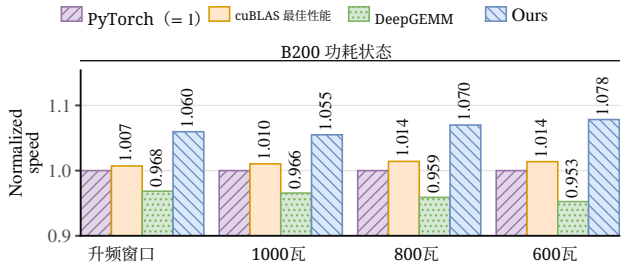


图 9：各形状的加速比 （%），我们的通用矩阵乘法相较于 cuBLAS 最优结果。我们在 300 种形状中的 294 种上超越了最先进水平 .

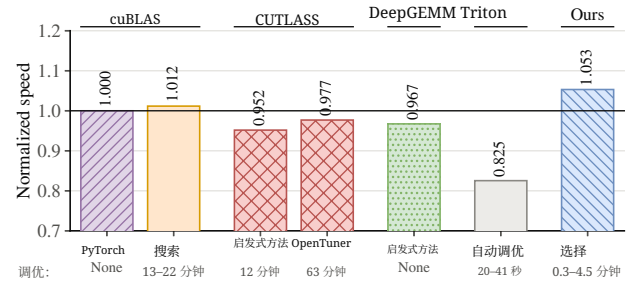


（a）图9中的各种形状

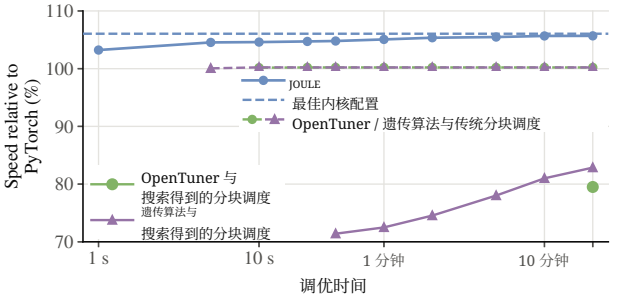


（b）B200 在不同功耗上限下

图10：相对于 PyTorch 归一化的几何平均速度。在（b）中，针对每个功耗上限，分别选取 cuBLAS 的最佳结果和我们的调度方案。



（a）每个调优器在其各自的设计空间中



（b）在我们的设计空间中的调优轨迹

图11：B200 上 15M = 8192 个形状的 MXFP8 GEMM 几何平均速度，相对于 PyTorch 归一化。

芯片局部内存放置的影响。 由于 cuBLAS 不会像第 4.1.3 节所述那样，将操作数放置到芯片局部物理内存中，因此我们评估了 B200 上的通用矩阵乘法，分别测试调用方进行芯片放置和不进行放置的情况。即使在默认的页面交错分配方式下、不进行放置，我们的内核在 60 种形状中的 58 种上仍优于 cuBLAS 最佳性能，几何平均值为 4.25%。启用芯粒本地放置后，分区操作数的跨芯片流量得以消除，几何平均加速比提升至 4.94%，实现了理论双芯片上限增益的 ~49%。

升频与功耗受限的情形。从空闲状态切换至满载后的最初几百毫秒内，B200 允许瞬时功耗升至约 1100W，因此每个内核比稳态时快 13%–21%。测量结果显示，此升频窗口的运行频率约为 1393 兆赫。内核在更高的有效功耗上限下运行，此时运行时间仍由能量决定。在图10b所示的所有功耗上限下，对于 30 种形状，我们的几何平均值始终优于所有基线：比 cuBLAS 最佳结果快 4.4%–6.4%，比 PyTorch 快 5.5%–7.8%，比 DeepGEMM 快 9.3%–13.2%。

5.3.2 设计空间、调度器与内核选择评估。

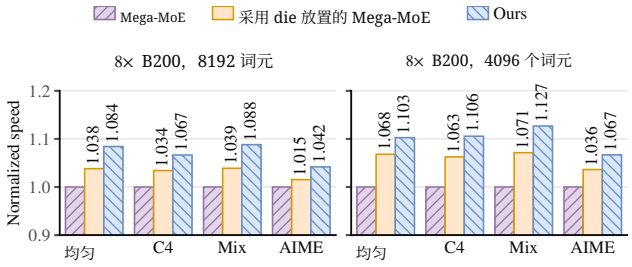


图12：Mega-MoE 在 8 张 B200 卡上运行 DeepSeek-V4-Pro 的速度。

设计空间评估。 图 11a 表明，调优器的性能受其内核所暴露的设计空间限制。现有的库和编译器仅搜索内核配置。由于这些固定映射无法解决跨芯片流量、二级缓存重读和尾波不平衡问题，即使在 cuBLAS 或 CUTLASS 中进行 13–63 分钟的设备端穷举搜索，性能也只能停留在 +1.2%（相较于 PyTorch 默认配置）的水平。通过在我们的设计空间中暴露每瓦片宽度、SM 对映射和执行顺序，性能上限提升至 +6.1%，并使我们的内核选择仅凭 CPU 预测即可达到 +5.0%。

调度器与内核选择对比通用调优器。图11b这表明，仅扩展设计空间而不进行分析式调度是不够的。在受限于传统分块调度时，OpenTuner 和遗传算法（GA）等通用调优器只搜索内核配置，性能最终停滞在+0.2%。相反，直接搜索原始瓦片调度时，两个通用调优器都要面对数千个相互耦合的每瓦片变量；缺乏协调的分配会破坏二级缓存局部性和跨 die 平衡，使其在进行 20 分钟 GPU 性能分析后仍停留在 −20.5% 和 −17.1%。此外，短时间的设备端计时运行会错误评估稳态功率受限性能，而完整的稳态性能分析则需要对每个候选项耗时数秒。

调度开销与选择质量。 对于一种配置，在单个 CPU 线程上，分析式调度可在 0.03–0.42 秒内完成。我们在一秒内达到 97.3%，并在 2 分钟内达到内核配置空间中最佳选择的 99.3%。

5.4 Mega-MoE 评估

我们评估第4.2 节中的融合 Mega-MoE 内核，其运行于 DeepSeek V4-Pro 上，在图 12 所示的四种路由分布下进行专家并行：来自混合语料库的真实轨迹、C4 [72], 和 AIME 提示词，以及均衡的均匀路由。在八个 B200 GPU 上，我们的内核相比采用裸片本地专家权重的上游 DeepGEMM 基线，性能最高提升 5.17%。加速几乎完全来自降低能量消耗，从而维持更高的受功率限制的时钟频率。在真实路由下每秩 8192 个词元时，周期数仅从 −0.5% 变为 +0.2%，而其流式多处理器频率则提升了 2.5%–4.6%。

5.5 PSoL* 差距分析与消融研究

*5.5.1 PSoL** 差距分析。图13将每个内核与 PSoL 的运行时间差距分解 * 为结构化损失和实现损失，

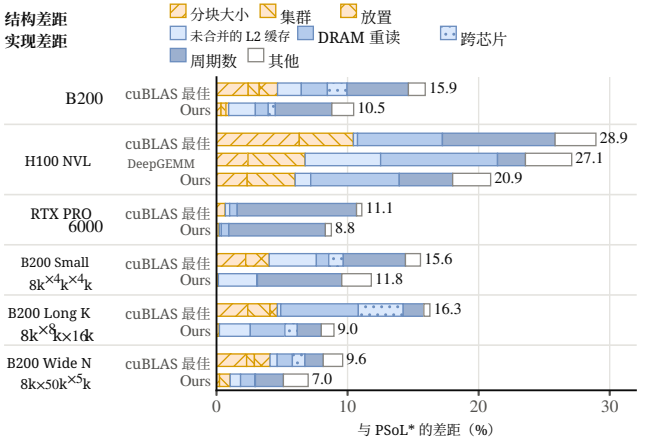


图 13：每个 GPU 以及三个具有代表性的 B200 形状上，60 种形状的几何平均值与 PSoL* a 之间的差距。我们的方法 GEMM 跨架构和瓶颈状态缩小了结构化损失和实现损失。

表2：在 60 种形状上，我们的方法 GEMM 相对于各基线在 SM 周期、每级传输字节数和能量方面的几何平均变化（%）。

我们的方法相对于	周期数	字节数	接收		二级缓存	跨芯片	DRAM	能量
B200, cuBLAS 最佳	−1.1	−19.4	−18.7	−16.0	−63.2	−22.7		−4.9
H100 NVL，cuBLAS 最佳	−10.1	−20.4	−24.6	−9.2	–	−1.2		−6.1
H100 NVL，DeepGEMM	+0.7	−15.0	−0.5	−39.8	–	−11.8		−5.1
RTX PRO 6000，cuBLAS 最优实现	−1.7	−3.5	−3.5	−3.6	–	+3.1		−2.1

而表2 总结了周期数、流量和能量方面的相应变化。

在 **B200 通用矩阵乘法**中，宽瓦片和芯粒本地放置将结构差距从 cuBLAS 最优实现的 4.7% 降至 0.9%，而瓦片调度器则将实现差距从 11.3% 降至 9.5%，使 PSoL* 达成率从 86.3% 提升至 90.5%。如表 2 所示，我们的通用矩阵乘法仅将流式多处理器周期减少了 1.1%，但将总数据移动量减少了 19.4%，能量消耗减少了 4.9%。在三种代表性形状中，主要损失从小型形状上的未合并的 L2 读取和额外周期，转变为长K形状上的 DRAM 和跨芯粒重复读取，而我们的设计同时解决了这两种情况。

在 **H100 NVL 通用矩阵乘法**中，我们的内核和 DeepGEMM 都使用256 × 128 瓦片来缩小相对于 cuBLAS 最优实现128 × 128 瓦片的结构差距；与此同时，在周期数几乎相同的情况下，我们的调度器将 L2 读取减少了 39.8%，将 DRAM 读取减少了 11.8%。剩余的实现差距源于大型K操作数超出一个 L2 半切片，以及无法同时对两个操作数进行 GPC 合并。

在**RTX PRO 6000 通用矩阵乘法**上，由于缺少 TMEM 和跨流式多处理器共享，两个内核都使用受寄存器限制的 256 × 128 瓦片，结构差距低于 1%，因此周期开销主导了性能差距。在不可分割形状上选择更窄的瓦片，并在工作集超过二级缓存容量时调整二级缓存遍历顺序，可将 PSoL* 达成率从 90.0% 提升至 91.9%。

表 3：Joule 各机制相对于 cuBLAS 最佳结果的消融研究。

配置	相对于 cuBLAS 最佳结果	能量（mJ）	周期数（M）
完整设计	+5.19%	381.0	0.485
– 旋转	+4.20%	385.3	0.493
– GPC 分组	+4.38%	384.3	0.484
– L2 驻留 A	+3.68%	386.2	0.483
– 芯粒本地放置 A	+2.26%	392.9	0.483
– 均匀瓦片分配	+1.75%	391.7	0.514
– 跨芯片输出拆分	+1.48%	396.2	0.475
cuBLAS 最佳	–	403.4	0.482

在**B200 Mega-MoE**上，八阶段权重流水线和 16 对齐的内核内瓦片化通过将每秩令牌数为 4096 时的步骤间空闲时间从 19% 降至 12%，缩小了实现差距。在采用均匀路由的单个 B200 上，我们的内核将每秩令牌数为 8192 时的 PSoL* 达成率从 81.3% 提升至 86.1%，并将每秩令牌数为 4096 时的达成率从 80.9% 提升至 84.7%。

5.5.2 消融研究。 为单独评估各项机制，表 3对我们在 B200 8192^3 mxfp8 GEMM 上的完整设计进行了消融。各个组件共同实现了 5.19% 的加速，其中跨芯片输出拆分、 A的芯粒本地放置以及 IP 负载均衡带来的收益最大。除均匀瓦片分配外，所有消融变体的周期数差异均在 3.7% 以内；在 P_{cap} 0.475M 下周期数最少的变体，却是最慢的，因为它能耗最高。各项单独损失相加后超过 5.19%，这是因为宽瓦片依赖旋转来隐藏尾声停顿，并依赖 IP 负载均衡来避免尾波空闲功耗。

6 局限性

功率封顶范围。 Joule 仅在内核运行于 P_{cap} 且数据移动消耗相当大一部分能量时，才能加速内核。低于 P_{cap} 的内核无法通过削减延迟隐藏流量获得时钟加速，而已接近数据移动下界的内核从 Joule 获得的收益甚微。逐 *GPU* 校准。 校准能量–频率模型需要为每种 GPU 架构执行一次微基准测试套件，并对内核运行进行性能分析，以测量电压曲线并拟合物理路径能耗。将该模型应用于同一 GPU 型号的另一块显卡，仍需拟合逐设备系数。输入数据分布。 必须根据目标输入数据分布校准模型，因为内核功耗取决于操作数值 [35, 37]。

7 相关工作

GPU 性能与功耗建模。 分析性能模型 [40, 43, 53, 55, 60] 和周期级模拟器 [7, 48, 82]在固定时钟频率下预测执行周期数，却忽略了运行频率取决于内核能耗这一事实。基于事件的 GPU 功耗模型 [6, 41, 47, 54, 56, 81, 86], 、支持动态电压频率调节的模型 [36], 和能耗屋脊线 [20, 45] 估算给定频率下的功耗。然而，它们忽略了现代物理数据路径，例如 GPC 请求合并和双芯片互连，忽略多域非线性电压缩放，也无法求解由 P_{cap}所允许的稳态时钟频率。系统级功耗封顶框架 [21, 23, 69, 70, 80, 94] 会调整批次大小、功耗限制或作业放置，而 Kuruppu 等人 [50] 观察到，在 H100 NVL 上减少流量会提高时钟频率，但它们都将内核实现视为固定不变。相比之下，Joule 对跨物理路径的每字节能耗成本以及非线性域电压曲线进行建模，直接求解 P_{cap}约束下的运行时间，实现近 1% 的能耗误差，并确立理论速度上限（PSoL* 和 PSoL(σ)），以指导内核设计（第 5.2节）。

GPU 张量核心库与多芯片局部性。 现代张量核心库和领域专用语言，例如 cuBLAS、CUTLASS、QuACK、Thun- derKittens 和 DeepGEMM [25, 28, 63, 65, 78],以及 FlashAt- tention [24, 74, 95] 和持久化巨型内核 [15, 19, 77],通过优化瓦片大小、集群多播和软件流水线，在 MMA 周期中隐藏内存延迟。由于它们将延迟隐藏后的流量视为零成本，这些设计让 B200 一半的 TMEM 在256 × 256 乒乓缓冲区中处于闲置状态，并且会导致二级缓存和 DRAM 读取冗余 P_{cap}。多芯片和 NUMA 感知型 GPU 研究 [5, 22, 42, 59]通过页面放置和瓦片映射减少远程流量，但没有考虑跨芯片速度不对称或功耗封顶频率节流。Joule 通过全 TMEM 宽瓦片设计、细粒度异步轮换和按速度比例进行双芯片划分弥补了这些不足，在稠密 GEMM 和融合 Mega-MoE 方面超越了最先进的库（第5.3节和5.4节）。

内核自动调优与调度搜索。 经验式自动调优器 [4, 62, 73, 84, 87, 89], 张量编译器 [17, 75, 91, 97], 和黑盒优化器 [1, 10, 27] 通过在硬件上对候选项计时，或评估基于周期的成本模型，在预定义的内核配置中进行搜索。然而，即使进行穷举搜索，它们的搜索空间也未涵盖细粒度的瓦片驻留顺序、可变宽度负载均衡和双芯片放置，从而限制了可实现的性能（第 5.3.2 节）。此外，短暂的设备端计时运行会以加速时钟频率执行，倾向于选择在.....后减速的调度方案 P_{cap} 进入稳态；而稳态硬件计时则要求每个候选项耗时数秒。 Joule 提供参数化设计空间，包括内核配置及瓦片调度，并配备 CPU 端瓦片调度器，通过整数规划合成瓦片调度；同时采用内核选择步骤，利用我们的功耗墙成本模型对候选项进行排序，无需 GPU 调优运行即可实现更高加速比。

8 结论

在功耗墙下，内核的运行时间由其能量消耗决定。我们提出了 Jo ule，这是一个利用能量–频率建模和功耗感知速度上限，指导宽瓦片异步内核与功耗感知调度超越延迟隐藏的框架。在 60 种稠密层形状上，这些设计相比穷举选出的 cuBLAS 算法，几何平均值分别提升了 4 200 、 6 94 、 64 100 、 2 12 6000 % （ B . % ）、 H NVL 上的 . %，以及 RTX PRO 上的 . %。采用 die 本地专家权重时，同样的原则使八张 B200 GPU 上的融合 Mega-MoE 相较于上游内核加速了 2.60%–5.17%。

我们计划将 Joule 开源，以支持研究社区，并鼓励功耗墙感知内核设计与调度。

致谢

文中使用了生成式人工智能工具来辅助写作和图表制作。作者已审阅全部内容，并对其承担全部责任。

参考文献

[1] Takuya Akiba、Shotaro Sano、Toshihiko Yanase、Takeru Ohta 和 Masanori Koyama。2019 年。Optuna：新一代超参数优化框架。载于第 25 届 ACM SIGKDD知识发现与数据挖掘国际会议（KDD）论文集。 2623 2631 10 1145 3292500 3330701 – 。doi: 。[2] / 。Hussam Al Daas 、 Grey Ballard 、 Laura Grigori 、 Suraj Kumar 和 Kathryn Rouse。2022 年。简短公告：与内存无关的并行矩阵乘法通信紧下界。载于第 34 届 ACM算法与体系结构并行性研讨会（SPAA）论文集。445–448。doi:10.1145/3490148.3538552[3] 阿里巴巴集团。2026 年。阿里巴巴发布 Qwen3.8-Max：迄今规模最大、能力最强的旗舰模型。https://www.alibabagroup.com/document- 2021044032125272064。[4] Jason Ansel、Shoaib Kamil、Kalyan Veeramachaneni、Jonathan Ragan-Kelley、Jeffrey Bosboom、Una-May O'Reilly 和 Saman Amarasinghe。2014 年。OpenTuner：可扩展的程序自动调优框架。载于第 23 届并行体系结构与编译国际会议（PACT）论文集。303– 316。doi:10.1145/2628071.2628092[5] Akhil Arunkumar、Evgeny Bolotin、Benjamin Cho、Ugljesa Milic、Eiman Ebrahimi、Oreste Villa、Aamer Jaleel、Carole-Jean Wu 和 David Nellans。2017 年。MCM-GPU：实现持续性能扩展性的多芯片模块GPU。载于第 44 届年度计算机体系结构国际研讨会（ISCA）论文集。[6] Akhil Arunkumar、Evgeny Bolotin、David Nellans 和 Carole-Jean Wu。2019 年。理解多模块GPU能效的未来。载于 IEEE高性能计算机体系结构国际研讨会（HPCA）。[7] Ali Bakhoda、George L. Yuan、Wilson W. L. Fung、Henry Wong 和 Tor M. Aamodt。2009 年。使用详细的GPU模拟器分析 CUDA工作负载。载于 IEEE 系统与软件性能分析国际研讨会（ISPASS）。 163 174 10 1109 2009 4919648 – 。doi: 。 /ISPASS。 。Grey [8] Ballard、James Demmel、Olga Holtz 和 Oded Schwartz。2011 年。数值线性代数中的通信最小化。SIAM J. Matrix Anal. Appl. 32，3（ 2011 866 901 10 1137 090769156 ），– 。doi: 。 / Olivier Beaumont、[9] Vincent Boudet、Fabrice Rastello 和 Yves Robert。2002 年。将正方形划分为矩形：NP完全性与近似算法。Algorithmica 34，3（2002），217–239。[10] James Bergstra、Rémi Bardenet、Yoshua Bengio 和 Balázs Kégl。2011 年。超参数优化算法。载于神经信息处理系统进展（NeurIPS），第 24 卷。[11] Christin David Bose、Cesar Avalos Baddouh、Junrui Pan 和 Timothy G. Rogers。2026 年。何时以及为何使用 TMA：基于工作负载的 NVIDIA Hopper GPU张量内存加速特征分析。载于 IEEE工作负载特征分析国际研讨会（IISWC）。351–362。doi:10.1109/IISWC71020.2026.00039[12] Richard P. Brent。1973 年。无导数最小化算法。Prentice-Hall，Englewood Cliffs，新泽西州。[13] Martin Burtscher、Ivan Zecena 和 Ziliang Zong。2014 年。使用 K20内置传感器测量 GPU功耗。载于通用 GPU处理研讨会（GPGPU）论文集。 28 36 10 1145 2576779 2576783 – 。doi: 。 / 。[14] Anantha P. Chandrakasan、Samuel Sheng 和 Robert W. Brodersen。1992 年。低功耗 CMOS 数字设计。IEEE固态电路期刊 27，4（1992），473–484。doi:10.1109/4.126534[15] Li-Wen Chang、Wenlei Bao、Qi Hou、Chengquan Jiang、Ningxin Zheng、Yinmin Zhong、Xuanrun Zhang、Zuquan Song、Chengji Yao、Ziheng Jiang、Haibin Lin、Xin Jin 和 Xin Liu。2024 年。FLUX：通过内核融合在 GPU 上实现快速的基于软件的通信重叠。arXiv:2406.06858。[16] Kimbo Chen 和 Dylan Patel。2026 年。剖析 Nvidia Blackwell—— 张量核心、 PTX 指令、 SASS 、Floorsweep、Yield。SemiAnalysis，https://newsletter.semianalysis.com/p/dissecting-nvidia-blackwell-tensor。[17] Tianqi Chen、Thierry Moreau、Ziheng Jiang、Lianmin Zheng、Eddie Yan、Haichen Shen、Meghan Cowan、Leyuan Wang、Yuwei Hu、Luis Ceze、Carlos Guestrin 和 Arvind Krishnamurthy。2018 年。TVM：面向深度学习的自动化端到端优化编译器。载于第 13 届 USENIX操作系统设计与实现研讨会（OSDI）。578–594。[18] Tianqi Chen、Lianmin Zheng、Eddie Yan、Ziheng Jiang、Thierry Moreau、Luis Ceze、Carlos Guestrin 和 Arvind Krishnamurthy。2018 年。学习如何优化张量程序。载于《神经信息处理系统进展》（NeurIPS），第 31 卷。[19] Xinhao Cheng、Zhihao Zhang、Yu Zhou、Jianan Ji、Jinchen Jiang、Zepeng Zhao、Ziruo Xiao、Zihao Ye、Yingyi Huang、Ruihang Lai、Hongyi Jin、Bohan Hou、Mengdi Wu、Yixin Dong、Anthony Yip、Zihao Ye、Songting Wang、Wenqin Yang、Xupeng Miao、Tianqi Chen 和 Zhihao Jia。2026 年。MPK：用于超大内核化张量程序的编译器和运行时。载于第 20 届 USENIX 操作系统设计与实现研讨会（OSDI）。1909–1926。arXiv:2512.22219。[20] Jee Whan Choi、Daniel Bedard、Robert Fowler 和 Richard Vuduc。2013 年。能量屋顶线模型。载于 IEEE 并行与分布式处理国际研讨会（IPDPS）。 661 672 10 1109 2013 77 – 。doi: 。/IPDPS。[21] 。Sangjin Choi 、 Inhoe Koo 、 Jeongseob Ahn 、 Myeongjae Jeon 和 Youngjin Kwon。2023 年。EnvPipe：节能且保持性能的 DNN 训练框架。载于 USENIX 年度技术会议（ATC）。[22] Euijun Chung 和 Hyesoon Kim。2026 年。用于多芯粒GPU上 GEMM 设计空间探索的快速局部性模拟器。arXiv:2606.11716。[23] Jae-Won Chung、Yile Gu、Insu Jang、Luoxi Meng、Nikhil Bansal 和 Mosharaf Chowdhury。2024 年。减少大模型训练中的能量浪费。载于 ACM SIGOPS 第 30 届操作系统原理研讨会（SOSP）论文集。144– 159。doi:10.1145/3694715.3695970[24] Tri Dao、Daniel Y. Fu、Stefano Ermon、Atri Rudra 和 Christopher Ré。2022 年。FlashAttention：具备 IO 感知能力的快速、内存高效精确注意力机制。载于《神经信息处理系统进展》（NeurIPS），第 35 卷。16344– 16359。[25] Dao-AILab。2026 年。QuACK：一组奇特的 CuTe内核。https://github. com/Dao-AILab/quack。[26] Bita Darvish Rouhani、Ritchie Zhao、Ankit More、Mathew Hall、Alireza Khodamoradi、Summer Deng、Dhruv Choudhary、Marius Cornea、Eric Dellinger、Kristof Denolf、Dusan Stosic、Venmugil Elango、Maximilian Golub、Alexander Heinecke、Phil James-Roxby、Dharmesh Jani、Gaurav Kolhe、Martin Langhammer、Ada Li、Levi Melnick、Maral Mesmakhosroshahi、Andres Rodriguez、Michael Schulte、Rasoul Shafipour、Lei Shao、Michael Siu、Pradeep Dubey、Paulius Mi- cikevicius、Maxim Naumov、Colin Verrilli、Ralph Wittig、Doug Burger 和 Eric Chung。2023 年。用于深度学习的微缩放数据格式。arXiv:2310.10537。[27] Kalyanmoy Deb、Amrit Pratap、Sameer Agarwal 和 T. Meyarivan。2002 年。一种快速且精英化的多目标遗传算法：NSGA-II。IEEE《进化计算汇刊》 6, 2 2002 182 197 10 1109 4235 996017 ( )，– 。doi: 。/ 。[28] 2025 8 。DeepSeek-AI。 。DeepGEMM：采用细粒度缩放的简洁高效 FP 通用矩阵乘法内核。https://github.com/deepseek-ai/DeepGEMM。[29] DeepSeek-AI。2026 年。DeepSeek-V4：迈向高效的百万 Token 上下文智能。arXiv:2606.19348。[30] Xiaoze Fan、Jianhao Wang、Weihao Cui、Han Zhao、Zhuobin Huang、Yangjie Zhou、Yuxian Qiu、Shixuan Sun、Bingsheng He、Quan Chen 和 Minyi Guo。2026 年。剖析分块缩放如何破坏 GPU细粒度调度。arXiv:2609.24270。[31] Matteo Frigo 和 Steven G. Johnson。2005 年。FFTW 的设计与实现 3 93, 2 2005 216 231 10 1109 2004 840301 。《IEEE 会议[32] 录》（ ），– 。doi: 。/JPROC。 。Trevor Gale、Deepak Narayanan、Cliff Young 和 Matei Zaharia。2023 年。MegaBlocks：采用专家混合的高效稀疏训练。载于《机器学习与系统论文集》（MLSys），第 5 卷。288–304。[33] G. H. Golub 和 V. Pereyra。1973 年。伪逆的微分与变量可分的非线性最小二乘问题。SIAM《数值分析杂志》10，2（1973），413–432。[34] Kazushige Goto 和 Robert A. van de Geijn。2008 年。高性能矩阵乘法剖析。ACM Trans. Math. Software 34，3（2008），1–25。doi:10.1145/ 1356052.1356053[35] Theo Gregersen、Pratyush Patel 和 Esha Choukse。2024 年。GPU 中与输入相关的功耗。载于高性能计算、网络、存储与分析国际会议（SC24-W）研讨会（可持续超级计算）。[36] João Guerreiro、Aleksandar Ilic、Nuno Roma 和 Pedro Tomás。2019 年。跨域动态电压频率调节的GPU功耗建模与解耦。IEEE《并行与分布式系统汇刊》30，11（2019）。[37] Horace He。2024 年。奇怪的是，向 GPU 提供"可预测"数据时，矩阵乘法运行得更快！https://www.thonking.ai/p/strangely-matrix- multiplications。[38] Arthur E. Hoerl 和 Robert W. Kennard。1970 年。岭回归：针对非正交问题的有偏估计。Technometrics 12，1（1970），55–67。doi:10.1080/00401706.1970.10488634[39] Jia-Wei Hong 和 H. T. Kung。1981 年。I/O复杂度：红蓝石子游戏。载于第 13 届 ACM 理论计算年度研讨会（STOC）论文集。326–333。[40] Sunpyo Hong 和 Hyesoon Kim。2009 年。具备内存级和线程级并行感知能力的 GPU 架构解析模型。载于第 36 届年度国际计算机体系结构研讨会（ISCA）论文集。[41] Sunpyo Hong 和 Hyesoon Kim。2010 年。集成式 GPU 功耗与性能模型。载于第 37 届年度国际计算机体系结构研讨会（ISCA）论文集。

[42] William Hu、Drew Wadsworth、Sean Siddens、Stanley Winata、Daniel Y. Fu、Ryann Swann、Muhammad Osama、Christopher Ré 和 Simran Arora。2025 年。HipKittens：快速而强劲的 AMD 内核。arXiv:2511.08083。[43] Je n-Cheng Huang、Joo Hwan Lee、Hyesoon Kim 和 Hsien-Hsin S. Lee。2014 年。GPUMech：基于区间分析的 GPU 性能建模技术。载于第 47 届 IEEE/ACM 国际微体系结构研讨会（MICRO）论文集。[44] Q. Huangfu 和 J. A. J. Hall。2018 年。并行化对偶修正单纯形法。数学规划计算 10，1（2018），119–142。doi:10.1007/s12532-017-0130-5[45] Aleksandar Ilic、Frederico Pratas 和 Leonel Sousa。2017 年。超越 Roofline：面向多核处理器的缓存感知功耗与能效建模。IEEE《计算汇刊》66，1（2017），52–58。[46] Dror Irony、Sivan Toledo 和 Alexander Tiskin。2004 年。分布式内存矩阵乘法的通信下界。《并行与分布式计算杂志》64，9（2004），1017–1026。[47] Vijay Kandiah、Scott Peverelle、Mahmoud Khairy、Junrui Pan、Amogh Manjunath、Timothy G. Rogers、Tor M. Aamodt 和 Nikos Hardavellas。2021 年。AccelWattch：面向现代 GPU 的功耗建模框架。载于第 54 届 IEEE/ACM 国际微体系结构研讨会（MICRO）论文集。[48] Mahmoud Khairy、Zhesheng Shen、Tor M. Aamodt 和 Timothy G. Rogers。2020 年。Accel-Sim：用于经验证 GPU 建模的可扩展仿真框架。载于第 47 届年度国际计算机体系结构研讨会（ISCA）论文集。[49] Michael Jaemin Kim、DoeHyun Yoon、Jovan Stojkovic、Gefei Zuo、JongBin Lim、Harsha Jagannati、Zhan Shu 和 Shobhit Kanaujia。2026 年。功耗屋顶线。https: //jovans2.github.io/files/Power_Roofline_MICRO.pdf。[50] Supun Kuruppu、Pranav Dangi、Wei Li、Zhenyu Bai 和 Tulika Mitra。2026 年。揭示物理非理想因素对内核性能的影响。arXiv:2610.37466。[51] Chester Lam。2025 年。Nvidia 的 B200：让 CUDA 这台巨型战车继续前进。Chips and Cheese, https://chipsandcheese.com/p/nvidias-b200-keeping-the- cuda-juggernaut。[52] Charles L. Lawson 和 Richard J. Hanson。1995 年。求解最小二乘问题。应用数学经典丛书，第 15 卷。工业与应用数学学会。doi:10.1137/1.9781611971217[53] Jounghoo Lee、Yeonan Ha、Suhyun Lee、Jinyoung Woo、Jinho Lee、Hanhwi Jang 和 Youngsok Kim。2022 年。GCoM：用于现代 GPU 精确解析建模的详细 GPU 核心模型。载于第 49 届年度国际计算机体系结构研讨会（ISCA）论文集。424–436。doi:10.1145/3470496.3527384[54] Kyungmi Lee、Zhiye Song、Eun Kyung Lee、Xin Zhang、Tamar Eilam 和 Anan- tha P. Chandrakasan。2026 年。EnergAIzer：面向人工智能工作负载的快速、准确 GPU 功耗估算框架。载于 IEEE 系统与软件性能分析国际研讨会（ISPASS）。435–447。doi:10.1109/ISPASS69572. 2026.00049[55] Seonho Lee、Amar Phanishayee 和 Divya Mahajan。2025 年。预测深度学习训练与推理的 GPU 性能。载于第 30 届 ACM 架构支持编程语言与操作系统国际会议（ASPLOS）论文集。493–508。doi:10.1145/3669940.3707265[56] Jingwen Leng、Tayler Hetherington、Ahmed ElTantawy、Syed Gilani、Nam Sung Kim、Tor M. Aamodt 和 Vijay Janapa Reddi。2013 年。GPUWattch：实现通用 GPU 中的能效优化。载于第 40 届年度国际计算机体系结构研讨会（ISCA）论文集。[57] Weile Luo、Ruibo Fan、Zeyu Li、Dayou Du、Hongyuan Liu、Qiang Wang 和 Xiaowen Chu。2026 年。通过微基准测试和多层级分析剖析 NVIDIA Hopper 架构。ACM《计算机系统汇刊》（2026）。doi:10.1145/3856813[58] Paulius Micikevicius、Dusan Stosic、Neil Burgess、Marius Cornea、Pradeep Dubey、Richard Grisenthwaite、Sangwon Ha、Alexander Heinecke、Patrick Judd、John Kamalu、Naveen Mellempudi、Stuart Oberman、Mohammad Shoeybi、Michael Siu 和 Hao Wu。2022 年。用于深度学习的 FP8 格式。arXiv:2209.05433。[59] Ugljesa Milic、Oreste Villa、Evgeny Bolotin、Akhil Arunkumar、Eiman Ebrahimi、Aamer Jaleel、Alex Ramirez 和 David Nellans。2017 年。超越处理器插槽：NUMA 感知 GPU。载于第 50 届 IEEE/ACM 国际微体系结构研讨会（MICRO）论文集。123–135。doi:10.1145/3123939.3124534[60] Zhiwen Mo、Yu Cheng、Lei Wang、Zhengju Tang、Lei Xu、Guoyu Li、Yuqi Dong、Lingxiao Ma、Yuqing Xia、Jilong Xue、Fan Yang、Luo Mai、Zhi Yang、Wayne Luk 和 Hongxiang Fan。2026 年。TileSight：首个基于第一性原理、以瓦片为中心、覆盖从核心到集群的 GPU 解析性能模型。arXiv:2607.22432。[61] Deepak Narayanan、Mohammad Shoeybi、Jared Casper、Patrick LeGresley、Mostofa Patwary、Vijay Korthikanti、Dmitri Vainbrand、Prethvi Kashinkunti、Julie Bernauer、Bryan Catanzaro、Amar Phanishayee 和 Matei Zaharia。2021 年。使用 Megatron-LM 在 GPU 集群上进行高效的大规模语言模型训练。载于高性能计算、网络、存储与分析国际会议（SC）论文集。1 15 10 1145 3458817 3476209 – 。doi：。/ 。[62] Cedric Nugteren 和 Valeriu Codreanu。 。2015 。CLTune：用于 OpenCL 内核的通用自动调优器。载于 IEEE 第九届嵌入式国际研讨会多核/众核片上系统（MCSoC）。195–202。doi:10.1109/MCSoC。2015.10[63] NVIDIA。2026。cuBLAS 库文档。https://docs.nvidia.com/cuda/ cublas/。[64] NVIDIA。2026。CUDA 编程指南：局部性域。https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/locality-domains.html。[65] NVIDIA。2026。CUTLASS：用于高性能线性代数的 CUDA 模板和 Python 领域特定语言。https://github.com/NVIDIA/cutlass。[66] NVIDIA。2026。Nsight Compute 性能分析指南。https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html。[67] Auguste Olivry、Julien Langou、Louis-Noël Pouchet、P. Sadayappan 和 Fabrice Rastello。2020。仿射程序参数化数据移动下界的自动推导。载于第 41 届 ACM SIGPLAN 编程语言设计与实现会议（PLDI）论文集。808–822。doi:10.1145/3385412.3385989[68] Junrui Pan、Weili An、Cesar Avalos Baddouh、Christin David Bose、Ni Kang、Aaron Barnes、Ahmad Alawneh、Fangjia Shen、Yechen Liu、Anusuya Nallathambi、Atthin Chandrashekar 和 Timothy G. Rogers。2026。面向人工智能时代的下一代异步分布式 GPU 架构设计。arXiv:2608.22602。[69] Pratyush Patel、Esha Choukse、Chaojie Zhang、Íñigo Goiri、Brijesh Warrier、Nithish Mahalingam 和 Ricardo Bianchini。2024。刻画云端大语言模型的功耗管理机会。载于第 29 届 ACM 国际体系结构支持编程语言与操作系统会议（ASPLOS）论文集。207–222。doi:10.1145/3620666.3651329[70] Pratyush Patel、Esha Choukse、Chaojie Zhang、Aashaka Shah、Íñigo Goiri、Saeed Maleki 和 Ricardo Bianchini。2024。Splitwise：利用阶段拆分实现高效的生成式大语言模型推理。载于第 51 届年度国际计算机体系结构研讨会（ISCA）。[71] PyTorch。2024。深入解析 CUTLASS 乒乓 GEMM 内核。PyTorch 博客, https://pytorch.org/blog/cutlass-ping-pong-gemm-kernel/。[72] Colin Raffel、Noam Shazeer、Adam Roberts、Katherine Lee、Sharan Narang、Michael Matena、Yanqi Zhou、Wei Li 和 Peter J. Liu。2020。探索统一文本到文本转换器的迁移学习极限。《机器学习研究杂志》21，140（2020），1–67。[73] Richard Schoonhoven、Bram Veenboer、Ben van Werkhoven 和 K. Joost Batenburg。2022。走向绿色：通过模型引导的自动调优优化 GPU 能效。载于 IEEE/ACM 高性能计算机系统性能建模、基准测试与仿真国际研讨会（PMBS）。[74] Jay Shah、Ganesh Bikshandi、Ying Zhang、Vijay Thakkar、Pradeep Ramani 和 Tri Dao。2024。FlashAttention-3：借助异步和低精度实现快速且准确的注意力机制。载于神经信息处理系统进展（NeurIPS）。[75] Junru Shao、Xiyou Zhou、Siyuan Feng、Bohan Hou、Ruihang Lai、Hongyi Jin、Wuwei Lin、Masahiro Masuda、Cody Hao Yu 和 Tianqi Chen。2022。利用概率程序进行张量程序优化。载于神经信息处理系统进展（NeurIPS），第 35 卷。35783–35796。[76] Tyler Michael Smith、Bradley Lowery、Julien Langou 和 Robert A. van de Geijn。2017。矩阵乘法的紧输入/输出下界。arXiv:1702.02017。[77] Benjamin Spector、Jordan Juravsky、Stuart Sul、Owen Dugan、Dylan Lim、Dan Fu、Simran Arora 和 Chris Ré。2025。看好了，没有气泡！为 Llama-1B 设计低延迟大型内核。Hazy Research 博客, https://hazyresearch.stanford.edu/blog/2025-05-27-no-bubbles。[78] Benjamin F. Spector、Simran Arora、Aaryan Singhal、Arjun Parthasarathy、Daniel Y. Fu 和 Christopher Ré。2025。ThunderKittens：简单、快速且可爱的内核。载于国际学习表征会议（ICLR）。2097–2130。[79] Philip B. Stark 和 Robert L. Parker。1995。有界变量最小二乘法：算法及其应用。《计算统计学》10（1995），129–141。[80] Jovan Stojkovic、Chaojie Zhang、Íñigo Goiri、Josep Torrellas 和 Esha Choukse。2025。DynamoLLM：面向性能和能效的大语言模型推理集群设计。载于 IEEE 高性能计算机体系结构国际研讨会（HPCA）。1348 1362 10 1109 61900 2025 00102 – 。doi:。/ HPCA 。 。[81] Bo Su、Junli Gu、Li Shen、Wei Huang、Joseph L. Greathouse 和 Zhiying Wang。2014。PPEP：在线性能、电源与能耗预测框架及动态电压频率调节空间探索。载于第 47 届 IEEE/ACM 国际微体系结构研讨会（MICRO）论文集。445–457。[82] Yifan Sun、Trinayan Baruah、Saiful A. Mojumder、Shi Dong、Xiang Gong、Shane Treadway、Yuhui Bao、Spencer Hance、Carter McCardwell、Vincent Zhao、Harrison Barclay、Amir Kavyan Ziabari、Zhongliang Chen、Rafael Ubal、José L. Abellán、John Kim、Ajay Joshi 和 David Kaeli。2019。MGPUSim：实现多 GPU 性能建模与优化。载于第 46 届国际计算机体系结构研讨会（ISCA）论文集。197–209。doi:10.1145/3307650.3322230[83] H. Theil 和 A. S. Goldberger。1961。经济学中的纯统计估计与混合统计估计。《国际经济评论》2，1（1961），65–78。[84] Philippe Tillet、H. T. Kung 和 David Cox。2019。Triton：一种用于分块神经网络计算的中间语言和编译器。载于

第 3 届 ACM SIGPLAN 机器学习与编程语言国际研讨会（MAPL）。 10 19 10 1145 3315508 3329973 – 。doi: 。/ 。Stanimire [85] Tomov、Jack Dongarra 和 Marc Baboulin。 。面向混合 GPU 加速众核系统的稠密线性代数。Parallel Comput。 36, 5 6 2010 232 240 10 1016 2009 12 005 – （ ） , – 。 [86] doi: 。/j.parco。 。 。Brandon Tran、Matthias Maiterth、Woong Shin、Matthew D. Sinclair 和 Shivaram Venkataraman。2026。Wattchmen：监视功率监视器——高保真、灵活的 GPU 能耗建模。载于第 40 届 ACM 国际超级计算会议（ICS）论文集。 145 160 10 1145 3797905 3800514 – 。doi: 。/ 。 [87] Ben van Werkhoven。 。Kernel Tuner：一种搜索优化型 GPU 代码自动调优器。《Future Generation Computer Systems》90（2019），347–358。 doi:10.1016/j.future.2018.08.004[88] Field G. Van Zee 和 Robert A. van de Geijn。2015。BLIS：快速实例化 BLAS 功能的框架。《ACM Trans. Math. Software》41，3（ 2015 1 33 10 1145 2764454 ），– 。doi: 。/ R. Clint Whaley、Antoine Petitet 和 [89] Jack J. Dongarra。2001。软件自动经验优化与 ATLAS 项目。《Parallel Comput.》27，1–2（2001）， 3 35 10 1016 0167 8191 00 00087 9 – 。doi: 。/S - （ ） - Samuel [90] Williams、Andrew Waterman 和 David Patterson。2009。Roofline：面向多核架构的富有洞察力的可视化性能模型。《Commun. ACM》52，4（2009），65–76。[91] Mengdi Wu、Xinhao Cheng、Shengyu Liu、Chunan Shi、Jianan Ji、Man Kit Ao、Praveen Velliengiri、Xupeng Miao、Oded Padon 和 Zhihao Jia。2025。Mirage：面向张量程序的多级超优化器。载于第 19 届 USENIX 操作系统设计与实现研讨会（OSDI）。21–38。[92] An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv、Chujie Zheng、Dayiheng Liu、Fan Zhou、Fei Huang、Feng Hu、Hao Ge、Haoran Wei、Huan Lin、Jialong Tang、Jian Yang、Jianhong Tu、Jianwei Zhang、Jianxin Yang、Jiaxi Yang、Jing Zhou、Jingren Zhou、Junyang Lin、Kai Dang、Keqin Bao、Kexin Yang、Le Yu、Lianghao Deng、Mei Li、Mingfeng Xue、Mingze Li、Pei Zhang、Peng Wang、Qin Zhu、Rui Men、Ruize Gao、Shixuan Liu、Shuang Luo、Tianhao Li、Tianyi Tang、Wenbiao Yin、Xingzhang Ren、Xinyu Wang、Xinyu Zhang、Xuancheng Ren、Yang Fan、Yang Su、Yichang Zhang、Yinger Zhang、Yu Wan、Yuqiong Liu、Zekun Wang、Zeyu Cui、Zhenru Zhang、Zhipeng Zhou 和 Zihan Qiu。2025。Qwen3 技术报告。arXiv:2505.09388。[93] Zeyu Yang、Karel Adamek 和 Wesley Armour。2024。GPU 的准确便捷能耗测量：NVIDIA GPU 内置功率传感器的详细研究。载于高性能计算、网络、存储与分析国际会议（SC）论文集。1–17。doi:10.1109/SC41406。2024.00028[94] Jie You、Jae-Won Chung 和 Mosharaf Chowdhury。2023。Zeus：理解并优化 DNN 训练的 GPU 能耗。载于 USENIX 网络系统设计与实现研讨会（NSDI）。[95] Ted Zadouri、Markus Hoehnerbach、Jay Shah、Timmy Liu、Vijay Thakkar 和 Tri Dao。2026。FlashAttention-4：面向非对称硬件扩展的算法与内核流水线协同设计。arXiv:2603.05451。[96] Dan Zhao、Siddharth Samsi、Joseph McDonald、Baolin Li、David Bestor、Michael Jones、Devesh Tiwari 和 Vijay Gadepally。2023。面向人工智能的可持续超级计算：HPC 规模的 GPU 功率封顶。载于 2023 年 ACM 云计算研讨会（SoCC）论文集。588–596。doi:10.1145/3620678.3624793[97] Lianmin Zheng、Chengfan Jia、Minmin Sun、Zhao Wu、Cody Hao Yu、Ameer Haj-Ali、Yida Wang、Jun Yang、Danyang Zhuo、Koushik Sen、Joseph E. Gonzalez 和 Ion Stoica。2020。Ansor：为深度学习生成高性能张量程序。载于第 14 届 USENIX 操作系统设计与实现研讨会（OSDI）。863–879。