arXiv:2602.20204v1 [cs.PL] 22 Feb 2026

分析基于 MLIR 的 AI 内核编译器中的延迟隐藏与并行性

Javed Absar[1], Samarth Narang[2],和 Muthu Baskaran[21]
Qualcomm Technologies International, Ltd.[2]Qualcomm Technologies, Inc.{mabsar,samanara,muthub}@qti.qualcomm.com

摘要

边缘设备上的 AI 内核编译取决于编译器利用并行性并在层次化内存和显式数据移动的情况下隐藏内存延迟的能力，以及其他因素。本文介绍了一种基准测试方法及其相应结果，针对基于 MLIR 的编译流水线中的三种由编译器控制的机制：向量化（Vec）、跨硬件上下文的多线程（MT），以及利用乒乓式暂存缓冲区实现双缓冲（DB），以便让 DMA 传输与计算重叠。我们采用由 Triton/Inductor 生成的内核，提出一套消融阶梯，以区分 Vec、MT 和 DB 各自的贡献，并以 GELU 作为代表性激活函数内核，量化 MT 加速比随问题规模的变化。结果表明，向量化为带宽敏感型内核带来了主要性能提升；当调度开销得到摊销后，MT 可带来显著改善；而当数据传输能够与计算重叠时（即既非纯内存受限、也非纯计算受限的极端情况），DB 则能带来额外收益。

1 引言

内核优化难以自动化：手写内核仍然难以超越，但生产系统需要能够适配快速演进的架构，并且可移植、易维护的代码生成。在边缘 NPU 上，性能取决于层次化内存、显式的 DMA 管理的传输，以及如何调度工作，以便在数据传输进行期间让计算单元保持忙碌。

端到端模型结果固然重要，但往往会掩盖 究竟哪些 映射机制能够提升性能，以及 原因。因此，我们采用可复现、以内核为中心的评估方法，与 Triton/Inductor 风格的代码生成相一致，在受控的问题规模下对具有代表性的算子内核进行基准测试，以分别量化向量化（Vec）、多线程（MT）和双缓冲（DB）的影响 [1]。Vec 利用数据级并行；MT 通过将独立分块分配到不同的硬件上下文，利用循环级和区域级并行；DB 则通过将内存数据传输与计算重叠，减少停顿时间。

为将性能提升归因于特定机制，我们使用简单的消融阶梯报告结果：

$$\texttt{Scalar} \rightarrow \texttt{Vec} \rightarrow \texttt{Vec+MT} \rightarrow \texttt{Vec+MT+DB}.$$

在这一递进方案中，Vec 单独体现 SIMD 风格降级，Vec+MT 用于量化线程级加速比的增量，而 Vec+MT+DB 则评估在 Vec 和 MT 已经就位后，显式的延迟隐藏调度能否带来额外提升。除了报告结果，我们还描述了用于实现 MT 和 DB 的具体 MLIR IR 模式及编译通道结构，使该方法易于复现和扩展。

2 实现细节：多线程与双缓冲

我们的实现基于 MLIR [2]，并遵循一项设计原则：尽可能长时间地在结构化中间表示中表达意图，只有在编译器确定调度后才降级为运行时构造。这样既能提升可移植性，也让转换更易于验证。

2.1 多线程（多线程）

硬件模型。 目标神经网络处理器支持多线程向量执行，其中每个硬件线程都关联一个独立的向量上下文（例如，向量寄存器和谓词状态）。这使得同一内核的独立分块能够并发执行，前提是分块迭代空间可以在没有跨线程依赖的情况下进行划分。

两阶段多线程降级。 我们将多线程实现为两阶段流水线，先保留结构化并行性，再引入显式的分叉-汇合。首先，*Form-Virtual-Threads* 将已分块的内核（例如，linalg.generic）重写为使用 scf.forall 的显式并行形式。该遍在分块空间上采用基于规模的收益启发式，并选择一种分布策略（块与块循环），以便在范围不均时平衡工作负载。

其次，*Form-Async-Threads* 使用 MLIR 的 Async[3] 将 scf.forall 降级为分叉-汇合表示。每个数据块成为一个产生令牌的 async.execute 区域；令牌被收集到一个异步组中，async.await all 在后续依赖计算之前形成一个屏障。 -

中间表示中的典型分叉-汇合骨架。 生成的模式刻意设计得精简且规整，因此很容易降级为协程/任务运行时 [3]:

%group =async.create_group %N scf.for %tile = ... { %tok = async.execute { /* 数据块主体 */ async.yield } async.add_to_group % tok, %group } async.await_all %group

为何使用 Async（而非直接降级为多线程）。 将多线程保留为 Async 中结构化的分叉-汇合形式，可在后期降级前以声明式形式保持并行语义，同时仍能直接转换为运行时调度 [3]。

2.2 双缓冲（DB）

双缓冲是一种软件流水线策略，通过交替使用两个暂存缓冲区（乒和乓），使数据传输与计算重叠。该技术与模调度和流水线循环执行密切相关 [4, 5]。我们分两个阶段实现双缓冲，将流水线调度的构建与引入特定于目标平台的异步传输原语分开。

阶段 1：结构化流水线。 阶段 1 匹配由分块操作通常生成的单缓冲分块循环“标准形式”：

`memref.subview` → `memref.alloc` → `memref.copy`,

先执行计算，然后执行回写序列。基于这一结构，该 pass 构建显式的乒乓调度。它生成序言代码，将第一个数据块预取到乒缓冲区中，然后将主循环重构为两个交替执行的子内核。每个子内核依次执行：(i) 将下一个数据块预取到另一个缓冲区；(ii) 使用当前缓冲区进行计算；(iii) 在当前归纳变量处重新物化子视图，以重建回写操作。布尔切换标志在每次迭代中选择乒或乓；该 pass 还会附加轻量级属性作为锚点，以便下一阶段可靠地识别预取、计算和回写区域。

阶段 2：异步 DMA 集成。阶段 2 将同步复制替换为显式的异步 DMA 操作。预取路径被改写为 `memref.dma start`，并使用不同的乒/乓标签；同时`memref.dma wait`会紧接在计算之前插入，以确保数据块驻留在 TCM 中。回写复制也可以采用类似的处理方式，并且该通道会生成成对平衡的标签释放。通过分阶段进行转换，我们先建立正确的调度，然后再将其映射到目标传输接口，这与流水线循环中的经典编译器实践相一致 [4, 5]。

将双缓冲与多线程结合。双缓冲可以自然地与多线程结合。一旦 memref.dma wait 确保数据块驻留在 TCM 中，计算区域便可利用 scf.forall（虚拟线程）和/或降低后的分叉-汇合表示，并发执行多个子数据块。这正是我们用于实现向量+多线程+双缓冲阶梯级别的组合方式。

3 基准测试设置与方法

我们评估了两个有代表性的内核。第一个是形状为 $[64, 128 \times 128]$ 个元素的带宽导向的二维向量加法微基准测试，有助于了解在内存密集型场景中，向量化和延迟隐藏各自带来的影响。第二个是 GELU，这是一种常见的激活函数内核，可代表 Transformer 推理子图；我们使用 Triton 实现的 GELU 作为具体的内核实例。

对于向量加法微基准测试，我们评估消融阶梯中的 Scalar、Vec、Vec+MT 和 Vec+MT+DB 变体。对于 GELU，我们针对问题规模扫描，分别报告单线程和多线程执行下的端到端延迟，以揭示多线程开销摊销和扩展性。我们以微秒（μs）为单位报告延迟；在适用情况下，加速比定义为单线程时间与多线程时间之比。

4 结果

4.1 二维向量加法消融阶梯

图 1 总结了二维向量加法的消融阶梯。向量化带来了最显著的改进（132，479 μs $\rightarrow$ 3、210 μs、$\sim$41。3$\times$），这与面向带宽的内核相一致；该内核可立即受益于 SIMD 风格降级。多线程和双缓冲带来了较小但可测量的增量收益（3，210 μs $\rightarrow$ 3,000μs $\rightarrow$ 2、689 μs），这表明，在完成向量化后，剩余的性能提升空间主要来自减少同步开销，以及部分重叠传输与计算效应，而不是提高算术吞吐量。

4.2 GELU：单线程与多线程扩展性对比

图 2 展示了 GELU 在不同规模下的延迟，图 3则展示了相应的多线程加速比。多线程在所有测试规模下都提升了性能，且加速比随问题规模增大而上升；在 1,048,576 个元素时达到 $\sim$3。91$\times$（单线程为 12,947 μ 秒，多线程为 3,313 μ 秒）。这一趋势表明，固定

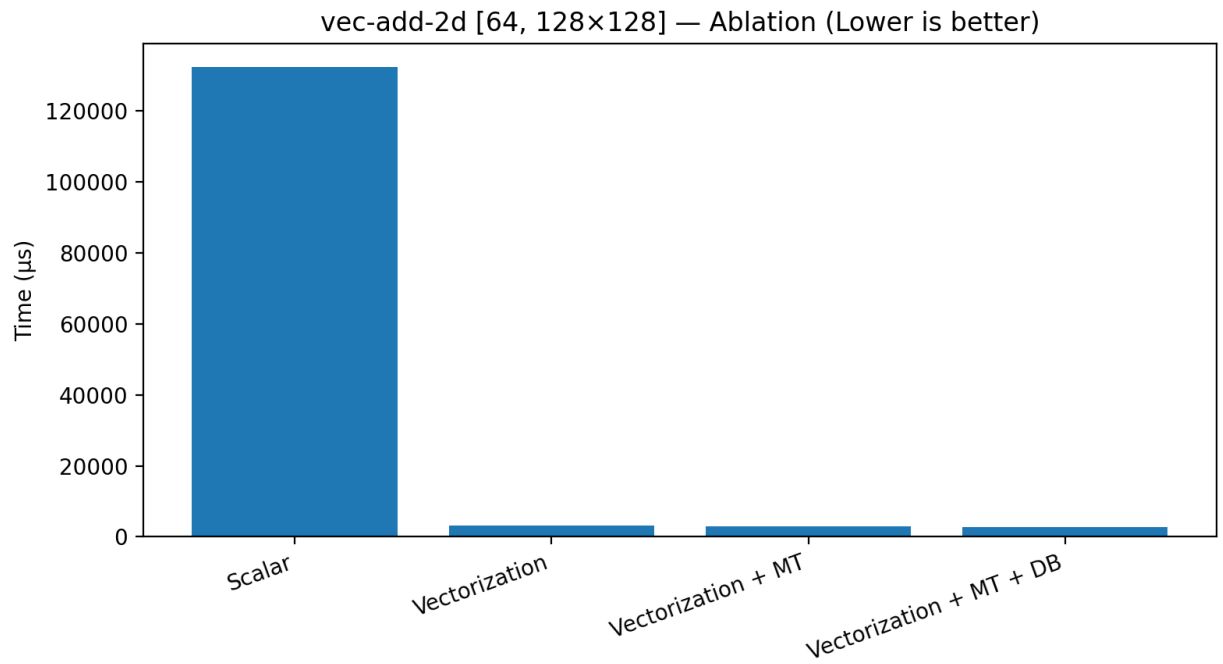

图 1：二维向量加法（$[64, 128 \times 128]$ 个元素）消融阶梯：标量（132,479 μs）、向量（3,210 μs）、向量+多线程（3,000 μs）、向量+多线程+双缓冲（2,689 μs）。

与分叉-汇合调度相关的开销会随着每个线程工作量的增加而得到摊销，而规模较大时出现的饱和现象则表明，内存带宽或屏障开销等共享瓶颈开始显现。

4.3 讨论

在这些内核中，向量化带来了最关键的第一阶性能提升，尤其是在带宽敏感场景下。当并行余量充足且内核规模足以摊销调度开销时，多线程还能带来显著的额外提升。当数据传输和计算都占有相当比重时，双缓冲可带来渐进式收益；在纯内存受限极限下，双缓冲受传输带宽制约，而在纯计算受限极限下，重叠机会有限。

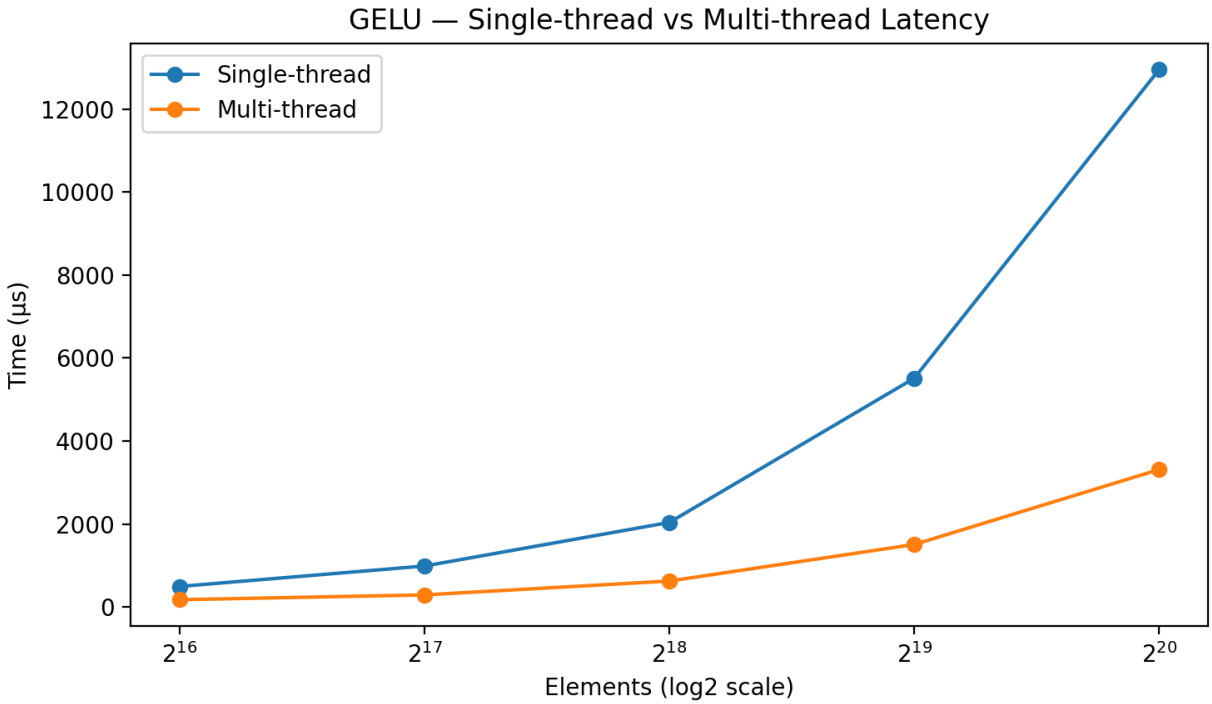

图2：单线程和多线程执行下 GELU 延迟与问题规模的关系。

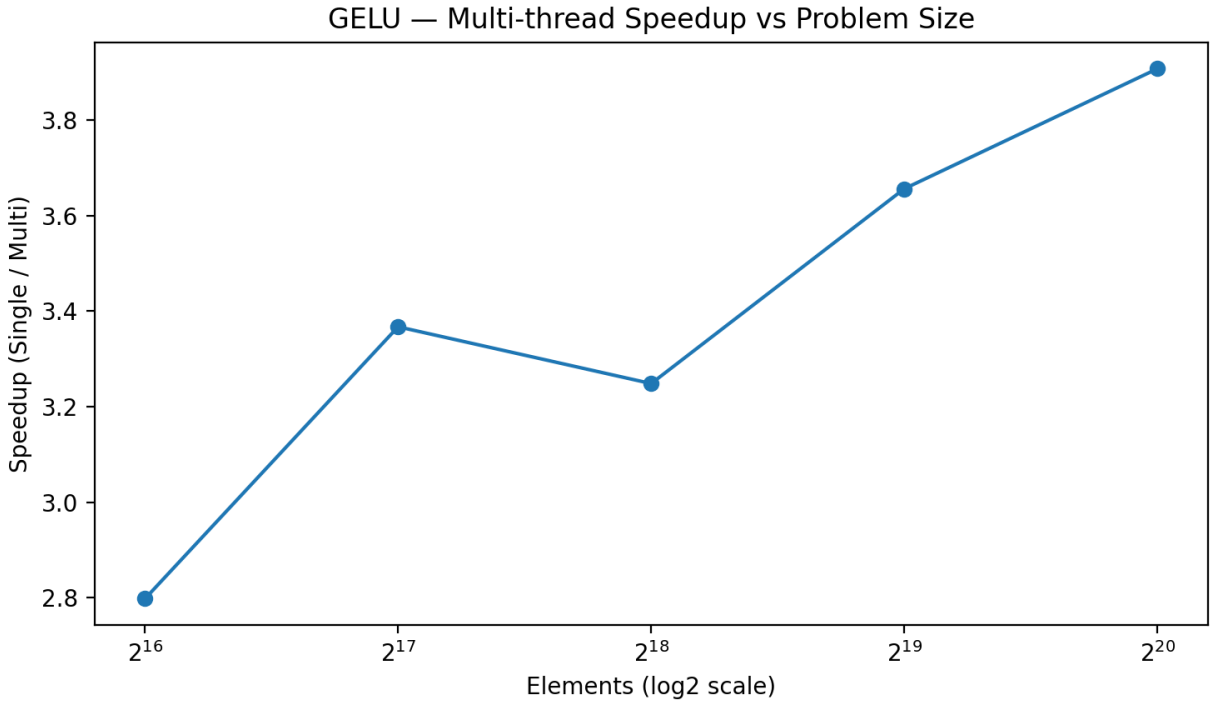

图3：GELU 多线程加速比（单线程/多线程）与问题规模的关系。

5 结论

我们提出了一套简洁且可复现的方法，用于分析基于 MLIR 的内核编译器中的向量化、多线程和双缓冲。通过这一逐级分析方法，可以更轻松地将性能提升归因于特定的编译器机制，而不是只报告端到端加速比。我们的测量结果表明，向量化是基础；开销得到摊销后，多线程可带来显著提升；当存在传输/计算重叠时，双缓冲还能进一步带来增益。未来的工作将扩展基准测试集（例如 RMSNorm、softmax），并将这些测量结果与预测模型相结合，以揭示重叠效率与 DMA 吞吐量及暂存存储器容量之间的关系。

参考文献

[1] Philippe Tillet、孔祥重和 David Cox。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论文集（*MAPL*），2019 年。[2] Chris Lattner、Mehdi Amini、Uday Bondhugula、Albert Cohen、Andy Davis、Jacques Pienaar、River Riddle、Tatiana Shpeisman、Nicolas Vasilache 和 Oleksandr Zinenko。MLIR：面向摩尔定律终结时代的编译器基础设施，2020 年。[3]LLVM项目。MLIR Async方言。https://mlir.llvm.org/docs/Dialects/AsyncDialect/，2026 年。访问日期：2026-01-21。[4] Monica S. Lam。软件流水：一种适用于超长指令字机器的有效调度技术。载于 *ACM SIGPLAN 1988* 编程语言设计与实现会议论文集（*PLDI*），1988 年。[5] B. Ramakrishna Rau。迭代模调度：一种用于循环软件流水的算法。第 *27* 届年度国际微体系结构研讨会论文集（*MICRO*），1994 年。