

通过在 IREE 中启用 RISC-V 微内核支持来加速 GenAI 工作负载

Adeel Ahmad, Ahmad Tameem Kamal, Nouman Amir, Bilal Zafar, Saad Bin Nasir

10xEngineers

摘要

该项目在 IREE (一个基于 MLIR 的机器学习编译器和运行时) 中实现了对 RISC-V 微内核的支持。该方法首先通过启用将 MLIR *linalg* 方言收缩操作降低为 *linalg.mmt4d* 操作来支持 RISC-V64 目标, 这是在 IREE 传递管道中进行的, 随后开发了针对 RISC-V 优化的微内核。性能提升与上游 IREE 和 *Llama.cpp* 进行了比较, 对比对象是 *Llama-3.2-1B-Instruct* 模型。

介绍

IREE (中间表示执行环境) [1], 一个基于 MLIR 的 [2] 编译器和运行时系统, 被开发用于将机器学习模型编译到各种平台, 如 CPU、GPU 和加速器。IREE 接受以 MLIR 代码形式的 ML 工作负载作为输入, 在其上应用经典的优化技术如操作符融合和切片, 并生成优化后的二进制文件供执行使用。尽管编译器生成的代码在大多数情况下表现合理, 但自定义内核在混合精度计算中表现更好。IREE 包括一个针对各种 CPU 和 GPU 架构进行了优化的小型内核库。IREE 包含针对 x86 和 ARM CPU 的小型内核, 然而, 尽管 RISC-V 在 AI 硬件领域中的存在不断增加, RISC-V 小型内核仍然缺失。这种限制导致基于 RISC-V 的硬件上 GenAI 模型的表现不佳。为了解决这个问题, 该项目使 IREE 支持 RISC-V 小型内核。

理论框架

矩阵乘法是 GenAI 工作负载中的关键计算。IREE 使用 MLIR *linalg* 方言收缩操作进行矩阵乘法, 这些操作在编译管道中会经过分割和融合处理。然而, 如果数据没有预先排列好, 则分块的矩阵乘法性能不佳, 导致高缓存未命中率 [3]。为了解决这个问题, IREE 利用了张量打包 MLIR 操作来重新排列数据, 确保在应用线性代数 *.mmt4d* 以进行优化计算之前, 分块连续存储在内存中。由线性代数 *.mmt4d* 生成的 4 维矩阵然后使用张量拆分操作转换回原始布局。

1. **张量打包**: 它将一个二维矩阵转换成一个四维的切片矩阵, 在这个四维矩阵中, 所有切片都连续

存储在内存中。

2. **线性代数 *.mmt4d***: 它执行由打包操作生成的左侧和右侧四维矩阵之间的矩阵乘法。这里的 ‘t’ 表示右侧矩阵的转置。
3. **张量拆分**: 它将由 *mmt4d* 产生的 4-D 结果矩阵转换回 2-D 布局。

这些 MLIR 操作被 IREE 编译过程降级为对微内核的调用。尽管 IREE 的架构 [4] 旨在使用户更轻松地集成任意微内核, 但目前它仅包含用于打包、解包和 *mmt4d* 操作的各种精度微内核, 适用于 x86 和 ARM64。对于 RISC-V 目标, 我们已经实现了 *mmt4d* 微内核。

方法论

所提出的方法可以被视为一个两步过程:

1. 第一部分涉及将 *linalg* 收缩操作转换为张量打包、张量解包和线性代数 *.mmt4d* 操作。当前, *linalg* 收缩操作转换为线性代数 *.mmt4d* (以及张量打包、张量拆分) 是在 x86-64 和 ARM64 目标架构的 *iree-codegen-materialize-device-encoding* 传递中进行的。此传递根据目标架构确定输入矩阵的 M 、 N 和 K 维度的瓦片大小。我们修改了此阶段, 以实现将收缩操作转换为线性代数 *.mmt4d*, 并对 RISC-V64 目标执行 **VLEN 感知切片**。一旦完成此转换, 线性代 *.mmt4d* 操作将通过后续阶段被转换为调用微内核函数。基于以下策略选择瓦片大小 [5]:

- (a) **预填**: 瓷砖大小 = $M, N, K=6, VLEN/8, 1$
- (b) **解码**: 瓷砖大小 = $M, N, K=1, VLEN/4, 1$

观察到，选择比这些更小的瓦片尺寸会导致硬件寄存器利用率不足，而使用更大的瓦片尺寸会增加寄存器压力，导致寄存器溢出和重新加载，并降低性能。

- 第二步包括实现微内核函数。mmt4d 微内核实现了 $f16 \times f16 \rightarrow f32$ 的情况，其中右侧和左侧操作数为 $f16$ 类型，结果操作数为 $f32$ 类型。分别针对大语言模型的预填充阶段和解码阶段实现了 mmt4d 微内核，因为预填充包含 GEMM 而解码阶段包含 GEMV 计算。

测试与性能基准测试

为了验证新实现的微内核的准确性，我们使用基于 **LM-评估框架** [6] 构建的框架评估了使用我们的微内核编译的 Llama-3.2-1B-Instruct 模型。结果总结在表 1 中。用 10x-IREE 编译的模型与从 Huggingface 获得的结果具有完全相同的分数。

Benchmark	Huggingface	10x-IREE
ARC_c	59.4%	59.4%
GPQA	27.2%	27.2%

表 1: LLaMA-3.2-1B-Instruct 模型在选定基准测试中的评估结果。该表比较了两个版本的模型性能：一个是从 Hugging Face 下载的，另一个是使用 10x-IREE 编译的。

为了性能基准测试，Llama-3.2-1B-Instruct 模型使用了 10x-IREE 进行编译，并记录了预填充和解码阶段的每秒标记数。结果总结在表 2 中。对于单线程运行，与上游 IREE 相比，我们在解码性能上观察到了 50 倍的提升。对于多线程运行，在预填充阶段观察到 2 倍的性能提升，在解码阶段观察到 17 倍的性能提升。

Phase	Threads	Llama.cpp	IREE	10x-IREE
预填充	1	0.04	0.14	0.18
	8	0.11	0.91	1.89
解码	1	0.03	0.02	0.99
	8	0.07	0.12	2.12

表 2: 性能（以每秒生成的标记数报告）在预填充和解码阶段，使用 llama.cpp、IREE 和 10x-IREE 编译的 LLaMA-3.2-1B-Instruct 模型。*

* 基准测试在配备 $1.66\text{GHz} \times 8$ RISC-V 向量核心的 MILK-V Jupiter 板上进行，VLEN=256 位，并采用 RVA22 配置文件。

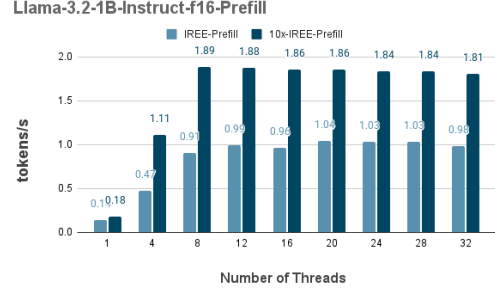


图 1: 预填充阶段性能比较在使用 IREE 编译的 Llama-3.2-1B-Instruct 和 10x-IREE 之间。*

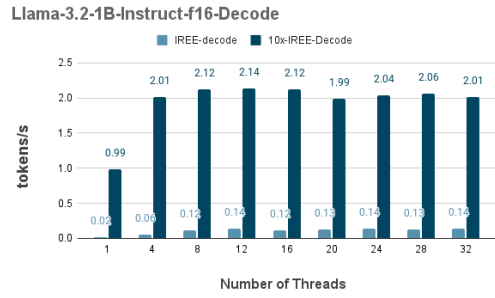


图 2: 解码阶段性能比较在使用 IREE 编译的 Llama-3.2-1B-Instruct 和 10x-IREE 之间。*

参考文献

- [1] “IREE,” accessed: Feb. 7, 2025. [Online]. Available: <https://iree.dev/>
- [2] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, “Mlir: A compiler infrastructure for the end of moore’s law,” 2020. [Online]. Available: <https://arxiv.org/abs/2002.11054>
- [3] “Matrix Multiplication with MMT4D,” accessed: Feb. 7, 2025. [Online]. Available: <https://iree.dev/community/blog/2021-10-13-matrix-multiplication-with-mmt4d/>
- [4] “IREE Source,” Github, accessed: Feb. 7, 2025. [Online]. Available: <https://github.com/iree-org/iree>
- [5] “LLM Optimization and Deployment on SiFive RISC-V Intelligence Products,” accessed: Feb. 7, 2025. [Online].

Available: <https://www.sifive.com/blog/llm-optimization-and-deployment-on-sifive-intellig>

- [6] “lm-evaluation-harness Source,” Github, accessed: Feb. 7, 2025. [Online]. Available: <https://github.com/EleutherAI/lm-evaluation-harness>