Accelerating GenAI Workloads by Enabling RISC-V Microkernel Support in IREE

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

[1]10xEngineers

Abstract

This project enables RISC-V microkernel support in IREE, a next-generation MLIR-based machine learning compiler and runtime. The approach begins by enabling the lowering of MLIR contraction ops to microkernel calls for the RISC-V target within the IREE pass pipeline, followed by the development of hand-optimized and vectorized microkernels tailored to RISC-V. The performance gains are compared with upstream IREE and Llama.cpp for the Llama-3.2-1B-Instruct model.

Introduction

As Generative AI (GenAI) models grow in size and complexity, optimized execution to efficiently utilize hardware capabilities has become essential. IREE (Intermediate Representation Execution Environment)[1], an MLIR-based[2] compiler and runtime, was developed to deploy machine learning models on various architectures, such as CPUs, GPUs, and accelerators. IREE accepts ML workload in the form of MLIR code as input, applies classical optimizations like operator fusion and tiling on it, and generates optimized binary for execution. While the compiler-generated code performs reasonably well in most cases, custom kernels perform better, particularly in mixed precision calculations. To harness the benefits of custom kernels, IREE includes a library of hand-optimized, vectorized microkernels for various CPU and GPU architectures. IREE includes microkernels for x86 and ARM CPUs, however, despite the increasing presence of RISC-V in the AI hardware space, RISC-V microkernels are still missing. This limitation results in poor performance of GenAI workloads on RISC-V-based hardware. To address this issue, this project focuses on enabling RISC-V microkernel support in IREE.

Theoretical Framework

Matrix multiplication is a critical computation in GenAI workloads. IREE uses MLIR linalg dialect contraction ops for matrix multiplication that undergoes tiling as it progresses through the compilation pipeline. However, tiled matmul incurs overhead if the data is not pre-arranged, leading to inefficient memory access and a high cache miss rate[3]. To address this, IREE utilizes ***tensor.pack*** MLIR operation to rearrange the data, ensuring that tiles are stored contiguously in memory before applying ***linalg.mmt4d*** for optimized computation. The 4-D matrix produced by *linalg.mmt4d* is then converted back to the original layout, using ***tensor.unpack*** operation.

1. ***tensor.pack*:** It takes a 2-D matrix and converts it into a tiled 4-D matrix in which all tiles are stored contiguously in memory.
2. ***linalg.mmt4d*:** It performs matrix multiplication between 4-D left-hand and right-side matrices, produced by the pack operations. The 't' in this stands for the transpose of the right-hand-side matrix.
3. ***tensor.unpack*:** It converts the 4-D result matrix produced by mmt4d back to 2-D layout.

These MLIR operations are lowered into calls to microkernels by IREE compilation passes. Even though IREE's architecture[4] is purposefully designed to make it easier for users to integrate arbitrary microkernels, currently it only contains microkernels for pack, unpack, and mmt4d operations, specialized for various precisions, for both x86 and ARM64. For the RISC-V target, we have implemented mmt4d microkernels.

Methodology

The proposed methodology can be viewed as a two step process:

1. The first part involves enabling transformations of linalg contraction ops to *tensor.pack*, *tensor.unpack* and *linalg.mmt4d* operations. Currently, the materialization of linalg contraction ops to *linalg.mmt4d* (and *tensor.pack*, *tensor.unpack*) is performed within the ***iree-codegen-materialize-device-encoding*** pass. This pass determines the tile sizes for the M, N, and K dimensions of the input matrices based on the target architecture e.g., x86, ARM. For RISCV, we modified this pass to enable the materialization of contraction ops into *linalg.mmt4d* and

*Corresponding author: adeel.ahmad@10xengineers.ai

to perform the **VLEN-aware tiling**. Once this materialization is complete, the *linalg.mmmt4d* operation would get transformed into calls to the generic microkernel functions via the subsequent passes. Tile sizes were selected based on the following strategy:

(a) **Prefill**: Tile Size= M,N,K=6,VLEN/4,1
(b) **Decode**: Tile Size=M,N,K=1,VLEN/8,1

It was observed that choosing a smaller tile size than these leads to underutilization of hardware registers, while using bigger tile sizes increases register pressure that causes register spills and reloads and degrades performance.

2. The second step includes enabling the selection of the specialized microkernel functions and implementing these functions. We implemented the selection of microkernel functions based on the tile sizes and data types of the operands. The mmt4d microkernels were implemented for the *f16*x*f16->f32* case, where rhs and lhs operands are of f16 type and the result operand is of type f32. Separate mmt4d microkernels were implemented for LLM's prefill and decode phases, because prefill has GEMM while the decode phase has GEMV computations.

Testing and Performance Benchmarking

To verify the accuracy of the newly implemented microkernels, we evaluated the Llama-3.2-1B-Instruct model compiled with our microkernels, using our framework built on top of **LM-Evaluation-Harness**[5]. The results are summarized in Table 1. The model compiled with 10x-IREE has exactly the same scores as the one obtained from Huggingface.

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

Table 1: *Evaluation results of the LLaMA-3.2-1B-Instruct model on selected benchmarks. The table compares the performance of two versions of the model: one downloaded from Hugging Face and the other compiled using 10x-IREE.*

For performance benchmarking Llama-3.2-1B-Instruct model was compiled using 10x-IREE, and tokens per second were recorded for prefill and decode phases. The results are summarized in Table 2. For the single-threaded run, we observed 50x gain in decode performance as compared to the upstream IREE. For multi-threaded run, a performance gain of 2x was observed in the prefill phase and 17x was observed in the decode phase.

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

Table 2: *Performance(reported as tokens per second) of the LLaMA-3.2-1B-Instruct model in prefill and decode stages, compiled using llama.cpp, IREE, and 10x-IREE. Results are reported for both 1-thread and 8-thread configurations. Benchmarking was conducted on a MILK-V Jupiter board featuring a 1.66GHz × 8 RISC-V vector cores system with VLEN=256 and RVA22.*

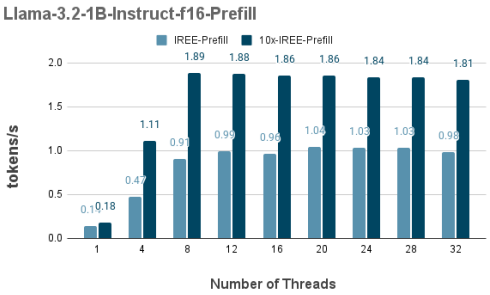

Figure 1: *Prefill phase performance comparison between Llama-3.2-1B-Instruct compiled with IREE and 10x-IREE. Benchmarking was conducted on a MILK-V Jupiter board featuring a 1.66GHz × 8 RISC-V vector cores system with VLEN=256 and RVA22.*

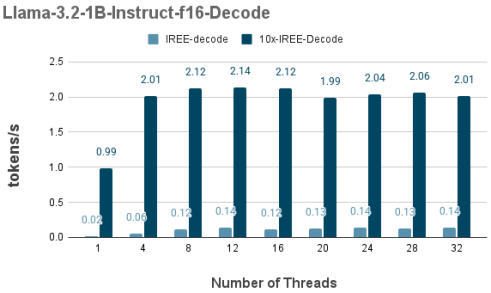

Figure 2: *Decode phase performance comparison between Llama-3.2-1B-Instruct compiled with IREE and 10x-IREE. Benchmarking was conducted on a MILK-V Jupiter board featuring a 1.66GHz × 8 RISC-V vector cores system with VLEN=256 and RVA22.*

How Would This Enhance the RISC-V Ecosystem?

This project enhances the performance of AI workloads compiled using IREE and establishes the foundation for integrating additional RISC-V microkernels within

the framework. It also serves as a generalized methodology, including algorithm design and implementation, that can be used as a template for writing hand-optimized and vectorized RISC-V kernels in other compiler frameworks and kernel libraries. Software support is essential for hardware adoption. Improved inference performance would shrink the gap between RISC-V and other architectures like x86 and ARM and incentivize chipmakers to develop RISC-V-based processors and accelerators. Moreover, this project aims to engage the RISC-V community in expanding RISC-V support in ML compilers and kernel libraries.

References

[1] *IREE*. Accessed: Feb. 7, 2025. Feb. 2025. URL: https://iree.dev/.

[2] Chris Lattner et al. *MLIR: A Compiler Infrastructure for the End of Moore's Law*. 2020. arXiv: 2002.11054 [cs.PL]. URL: https://arxiv.org/abs/2002.11054.

[3] *Matrix Multiplication with MMT4D*. Accessed: Feb. 7, 2025. Oct. 2021. URL: https://iree.dev/community/blog/2021-10-13-matrix-multiplication-with-mmt4d/.

[4] *IREE Source*. Github. Accessed: Feb. 7, 2025. Feb. 2025. URL: https://github.com/iree-org/iree.

[5] *lm-evaluation-harness Source*. Github. Accessed: Feb. 7, 2025. Feb. 2025. URL: https://github.com/EleutherAI/lm-evaluation-harness.