Accelerating Quantized LLM Inference for Embedded RISC-V CPUs with Vector Extension (RVV)

Yueh-Feng Lee, Yi-Jui Chu, Chih-Chung Huang, Heng-Kuan Lee

Andes Tech.

Abstract

This presentation focuses on optimizing the open-source llama.cpp project for the RISC-V vector extension. Specifically, we evaluate the performance of the LLM models running on an Andes AX45MPV RVV core implemented on FPGA. With VLEN 512, the 4-bit TinyLlama 1.1B model achieves a 21.7x speedup, and scaling results suggest that a single RVV core can achieve near real-time inference. Additionally, ongoing efforts are focused on optimizing smaller DeepSeek-related models to enhance efficiency on RISC-V hardware.

Introduction

The rapid advancement of large language models (LLMs) has transformed natural language processing, enabling sophisticated text understanding and generation. Among them, the open-source llama.cpp project has gained recognition for its efficiency and flexibility. Meanwhile, RISC-V has emerged as a powerful open ISA, well-suited for embedded systems.

This presentation explores the optimization of llama.cpp for the RISC-V V (RVV) extension [1] and evaluates its performance on a Andes RVV core using FPGA.

LLAMA CPP and Recent LLM Development

llama.cpp [2] is an open-source project for large language model inference and is written in C++ language. It is suitable for embedded systems and can be ported to bare-metal environment with not too much effort. Language models hosted by Hugging Face can be easily converted to its gguf format, which can be executed by the llama.cpp runtime. Various notable LLM models are already converted for llama.cpp. For instance, the widely used LLaMA 2 7B model [3], as well as the TinyLLaMA model [4], which is tailored for resource-constrained embedded devices.

The rise of Mixture of Experts (MoE) models offers a promising approach to improving inference efficiency for large models. MoE dynamically activates only a subset of expert networks per input, reducing computation while maintaining model capacity. Recent MoE architectures, such as DeepSeek [5], leverage sparse activation to achieve high efficiency. Notably, both DeepSeek and various distilled models are now available in gguf format and can be executed with llama.cpp.

LLAMA CPP with RISC-V V Extension

The upstream llama.cpp project includes RVV optimizations for quantized formats and has been reported to achieve up to a 3.5x speedup on a specific VLEN 256 real-chip platform. However, the upstream implementation is reported to contain bugs that prevent it from generating human recognizable tokens under VLEN 512, limiting its flexibility on higher-capacity vector configurations.

Optimization for Mixed Quantized and Floating-Point Computation

Matrix multiplication is the dominant operation in large language model inference. To compute quantized model in llama.cpp, matrix and vector computations are decomposed into quantized blocks, where each block represents a chunk of data with some quantization scale information. The default block size is 32, with smaller block sizes offering higher precision at the cost of increased model sizes.

To accelerate matrix multiplication for quantized blocks for larger VLENs, we rearrange data into an interleaved format, enhancing data reuse and improving efficiency for dot product-like instructions. Beyond quantized optimizations, we also optimize certain floating-point computation using RVV. Despite weights being stored as integer types in quantized models, floating-point computation remains integral to the inference process. Fig. 1 illustrates a partial computation graph of TinyLLaMA, revealing the intermixing of quantized and floating-point computations during execution.

Performance

Table 1 presents the inference performance of a 4-bit quantized TinyLLaMA 1.1B model measured by llama.cpp llama-bench, running on the AX45MPV FPGA. The AX45MPV core features VLEN 512 / DLEN 512, operating at 40 MHz, with performance results scaled to 1 GHz. The Andes RVV optimizations achieve a 21.7x speedup in token generation compared to the upstream scalar implementation.

Table 2 provides a breakdown of operation latencies within the TinyLLaMA model. In llama.cpp, the MUL_MAT operation includes both matrix multiplication and vector product computations. After Andes RVV optimization, matrix multiplication remains the dominant operation, while SOFT_MAX, currently computed as a

scalar operation, presents an opportunity for further optimization using RVV acceleration.

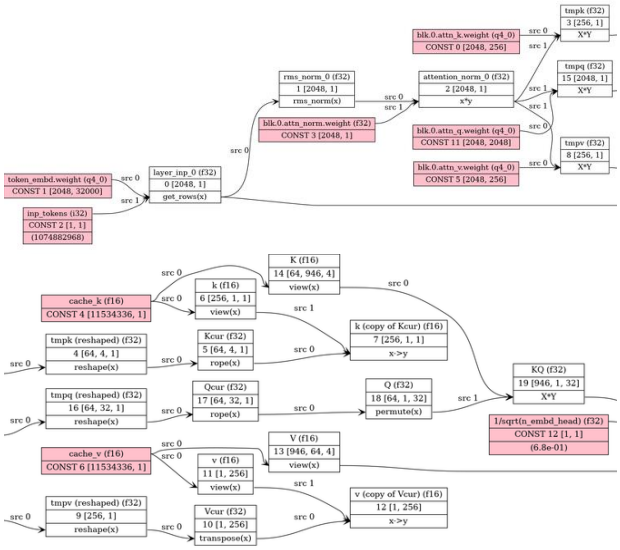

Fig 1. Partial computation graph of TinyLLaMA model in llama.cpp

Model	SW Option	tg128 (Tokens / Sec)	tg128 Scaleup 1GHz	Speedup
TinyLLaMA 1.1B	Scalar	0.0105	0.2625	1x
	Andes RVV	0.2279	5.6975	21.7x

Table 1. TinyLLAMA 1.1B 4-bit single-core performance scaled up to 1GHz

Table 3 shows the performance of smaller DeepSeek-related models run on an Andes scalar real-chip platform at 1.6 GHz with a smaller L2 cache. The scalar results indicate that DeepSeek v2 Lite Chat 16B outperforms LLaMA 2 7B because it is a Mixture of Experts (MoE) model, activating fewer parameters during inference. Additionally, DeepSeek v2 Lite Chat surpasses DeepSeek R1 Distill Qwen 1.5B, as the latter remains a dense model even after distillation. Table 4 shows the preliminary RVV optimization result of 4-bit DeepSeek v2 Lite Chat 16B.

Conclusion

Our results indicate that a single RVV core with VLEN 512 is expected to achieve near real-time inference for the 4-bit TinyLLaMA 1.1B model.

Optimization and performance evaluation are ongoing, and future updates will provide detailed results on optimizing smaller DeepSeek-related models using RVV. We anticipate that 4-bit DeepSeek v2 Lite Chat 16B could achieve near real-time inference with a small number of cores when optimized effectively using RVV.

Operation	Percentage
ADD	0.38%
MUL	0.91%
RMS_NORM	0.85%
MUL_MAT	80.42%
CPY	0.20%
CONT	0.07%
RESHAPE	0.06%
VIEW	0.12%
PERMUTE	0.06%
TRANSPOSE	0.03%
GET_ROWS	0.01%
SOFT_MAX	12.79%
ROPE	0.35%
UNARY	3.76%
Total	100%

Table 2. TinyLLaMA 1.1B 4-bit RVV op latency

Model	Model Type	tg128 (Tokens / Sec)	tg128 Scaledown 1GHz
TinyLLaMA 1.1B	Dense	0.4080	0.2550
LLaMA 2 7B	Dense	0.0632	0.0395
DeepSeek v2 Lite Chat 16B	MoE	0.1503	0.0939
DeepSeek R1 Distill Qwen 1.5B	Dense	0.1887	0.1173

Table 3. Smaller DeepSeek related 4-bit models run as scalar on real-chip platform

Model	SW Option	tg128 (Tokens / Sec)	tg128 Scaleup 1GHz
DeepSeek v2 Lite Chat 16B	Andes RVV	0.0437	1.0925

Table 4. Preliminary DeepSeek v2 Lite Chat 16B 4-bit single-core performance scaled up to 1GHz

References

[1] RISC-V "V" Vector Extension Specification. https://github.com/riscvarchive/riscv-v-spec

[2] llama.cpp. https://github.com/ggerganov/llama.cpp

[3] LLaMA 2. https://llama.meta.com/llama2/

[4] TinyLLaMA. https://github.com/jzhang38/TinyLlama

[5] DeepSeek. https://github.com/deepseek-ai