POSTER: V-Seek: Optimizing LLM Reasoning on A Server-Class General-Purpose RISC-V Platform

POSTER: V-Seek: Optimizing LLM Reasoning on A Server-Class General-Purpose RISC-V Platform

Javier Jesus Poveda Rodrigo
javier.poveda@polito.it
Politecnico di Torino
Turin, Italy

Mohamed Amine Hamdi
mohamed.hamdi@polito.it
Politecnico di Torino
Turin, Italy

Cyril Koenig
cykoenig@iis.ee.ethz.ch
ETH Zürich
Zurich, Switzerland

Alessio Burrello
alessio.burrello@polito.it
Politecnico di Torino
Turin, Italy

Daniele Jahier Pagliari
daniele.jahier@polito.it
Politecnico di Torino
Turin, Italy

Luca Benini
lbenini@iis.ee.ethz.ch
ETH Zürich
Zurich, Switzerland

Abstract

This paper addresses the problem of deployment of LLMs on RISC-V-based CPU systems by optimizing LLM inference on the Sophon SG2042. We evaluate the inference performance of two state-of-the-art LLMs optimised for reasoning: DeepSeek R1 Distill Llama 8B and DeepSeek R1 Distill QWEN 14B. Thanks to our optimizations on top of the `llama.cpp` inference library, we achieve token generation speeds of 4.32/2.29 tokens per second and prompt processing speeds of 6.54/3.68 tokens per second, with a significant speedup of up to 2.9×/3.0× compared to a direct porting of the same library.

CCS Concepts

• **Computer systems organization** → Heterogeneous (hybrid) systems; • **Computing methodologies** → Natural language processing.

Keywords

RISC-V, NLP, Inference Optimization, Many-core CPU

ACM Reference Format:
Javier Jesus Poveda Rodrigo, Mohamed Amine Hamdi, Cyril Koenig, Alessio Burrello, Daniele Jahier Pagliari, and Luca Benini. 2025. POSTER: V-Seek: Optimizing LLM Reasoning on A Server-Class General-Purpose RISC-V Platform. In *22nd ACM International Conference on Computing Frontiers (CF '25), May 28–30, 2025, Cagliari, Italy*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3719276.3727954

1 Introduction

Hyperscalers such as AWS and AI deployment companies like OpenAI typically accelerate Large Language Model (LLM) workloads using GPU clusters or specialized accelerators such as Tensor Processing Units (TPUs). Recently, however, many-core CPU architectures have emerged as promising alternatives, offering advantages such as reduced hardware costs and increased flexibility, particularly beneficial for on-premises deployments and low-latency edge servers. While existing research in this domain predominantly focuses on x86 and ARM architectures, exploration of many-core processors based on the open-source and flexible RISC-V Instruction Set Architecture (ISA) remains limited [1]. To address this gap, this paper adapts and optimizes a state-of-the-art LLM inference framework, `llama.cpp` [6], for the first commodity general-purpose, many-core RISC-V platform, the Sophon SG2042 [1]. Our evaluations using two recent open-source LLMs optimized for reasoning tasks—DeepSeek R1 Distill Llama 8B and QWEN 14B—demonstrate significant performance improvements over a baseline `llama.cpp` implementation. Specifically, we achieve up to 3.0× speedup in token generation and 2.8× speedup in prompt processing (prefill) at 4-bit precision, corresponding to throughputs of 4.32/2.29 tok/s and 6.54/3.68 tok/s, respectively. For the vanilla Llama 7B model, we attain throughputs of 6.63 tok/s for token generation and 13.07 tok/s for prompt processing, speeding up by 4.3× and 5.5× compared to the baseline. Our throughput is also 1.65× higher compared to the previously best-reported performance on the SG2042 [7] and competitive with CPU-based inference on established x86 architectures [3].

2 Methods

To explore optimization strategies for LLM inference on RISC-V server-class platforms, we focus on the MILK-V system, which consists of the 64-core SG2042 processor paired with 128GB of DRAM memory [1]. A block diagram of the platform is presented in Fig. 1-center. We identify three primary software optimization directions, inspired by prior work on other architectures [2, 4, 5].

I) We develop an optimized **kernel** for quantized GEMM/GEMV, i.e., the major bottleneck in LLM layers, fully exploiting hardware characteristics such as memory architecture, pipeline efficiency, and vector units. Its pseudocode is shown in Fig. 1-right. Initially, the `fp32` input (either a vector or thin matrix) is quantized to `int8`. Subsequently, a GEMV operation is performed using two nested loops, with the outer one iterating over the rows of the input matrix (A) and the inner loop over its columns. After completing the inner loop, de-quantization is executed using the scale factors from blocks of matrix A and matrix B, yielding a final `fp32` output. The two loops are unrolled to maximize data reuse and vector unit utilization.

II) We select and combine **compilation** toolchains to achieve advanced code optimizations while fully utilizing available ISA extensions. Specifically, GEMV/GEMM kernel compilation utilizes the Xuantie fork of GCC 10.4, as only this compiler supports the compilation for the vector units of the Sophon SG2042. Instead, for compiling the complete `llama.cpp` framework, we evaluate GCC

CF '25, May 28–30, 2025, Cagliari, Italy

ACM ISBN 979-8-4007-1528-0/2025/05
https://doi.org/10.1145/3719276.3727954

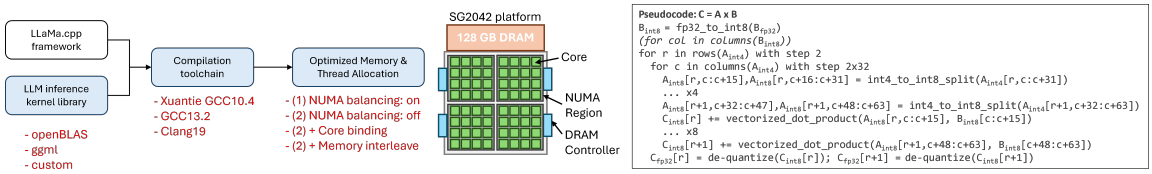

Figure 1: Left to right: optimization flow and contributions. SG2042 block diagram. Pseudocode of the proposed kernel.

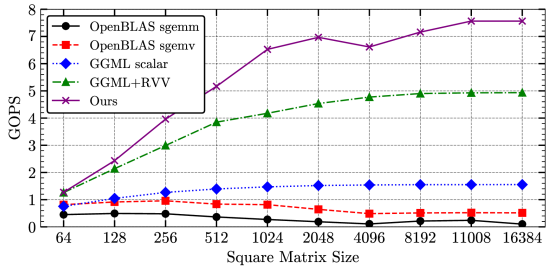

Figure 2: GEMV scalability experiments.

13.2 and Clang 19, given that Xuantie GCC 10.4 is incompatible with the latest version of `llama.cpp`.

III) We optimize the model **mapping**, particularly memory page/thread allocation, to effectively manage the memory hierarchy of this class of systems, with the goal of minimizing thread migrations among core regions and high-latency memory accesses (i.e., to far memory regions). To this end, we explore several configurations of *numactl*, combining four distinct policies: (a) NUMA Balancing enabled with all other options disabled, (b) all options disabled, (c) NUMA Balancing disabled with Core Binding enabled, and (d) NUMA Balancing disabled with Memory Interleaving enabled.

These optimizations are implemented within the `llama.cpp` framework [6] and evaluated across three open-source LLMs of varying sizes, each quantized using the Q4_0 method: vanilla Llama 7B, DeepSeek R1 Distill Llama 8B, and DeepSeek R1 Distill QWEN 14B (referred to as 7B, 8B, and 14B, respectively henceforth).

3 Results

To show the results of our optimization, we executed the prefill of the three LLMs with user prompt "Explain to me what is RISC-V, what are its principles and why it is so cool?" (22 tokens), while we averaged the token generation performance over 256 tokens.

Kernel Scaling. Fig.2 shows the single-thread scalability of kernels. Compared to the best baseline (GGML from llama.cpp + RVV extension), we improve the GOPS by +38.3% on average, peaking at +56.3% at matrix size 1024.

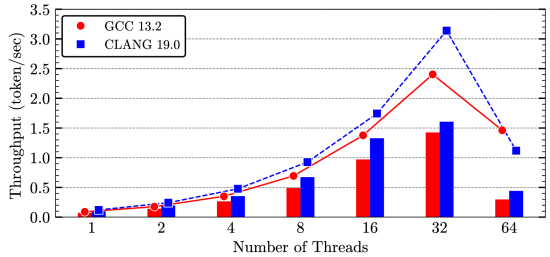

Figure 3: Compilers comparison. Bar=token gen, Line=prefill.

Compiler exploration. In Fig.3, using our kernel, we evaluate the compiler's impact on the 8B model inference. Clang 19 constantly outperforms GCC 13.2, reaching average performance gains

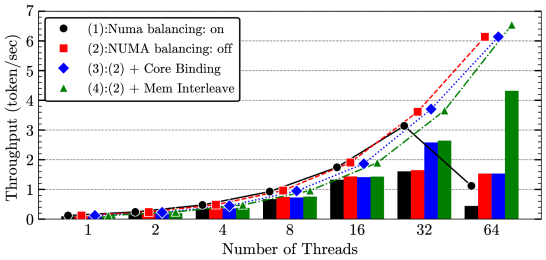

Figure 4: NUMA policies. Bar=token gen, Line=prefill.

Table 1: LLaMa2 7B performance comparison.

Source	Hardware	Inference Framework	Data format	Tok/s @2GHz	Efficiency (T/s/mW)
Baseline	SG2042	Pytorch	FP16	0.17	0.85
		Llama.cpp + OpenBLAS	FP16	0.55	5
			GGML Q4_0	1.55	13
PerfXLab [7]	SG2042	PerfXLM	AWQ INT4	4.01	33
QIGen* [3]	AMD EPYC 7742	PyTorch	4-bit: full	9.07	45
			4-bit: 128g	8.23	41
Ours	SG2042	Llama.cpp	GGML Q4_0	6.63	55

* throughput scaled at 2 GHz clock frequency.

of 34% and 25% for token generation and prefill, respectively. The crucial reason is the combination of ISA extension support, and more advanced compilation passes (e.g., more aggressive in-lining and loop unrolling). Regardless of the compiler used, using > 32 threads leads to a performance loss. This is attributed to the default NUMA balancing policy, which is suboptimal for LLM inference (due to the predictable nature of this workload) and leads to an unnecessarily high number of thread and memory page migrations.

NUMA policy impact. Fig.4 compares NUMA policies on the 8B model. With NUMA balancing off and memory interleaving on, given the lower number of thread migrations, we achieve the best results for both token generation (4.32 tok/s) and for prefill (6.54 tok/s), thanks to the strong reduction in memory page migration.

State-of-the-art. Overall, thanks to our optimizations, the 7B, 8B and 14B LLMs reach a maximum throughput of 13.07/6.54/3.68 tok/s, outperforming a baseline `llama.cpp` by up to 5.5×/2.9×/3×. Compared to the best-reported result on the SG2042 [7], we improve the peak throughput on LLama 7B by 1.65× (see Tab. 1). Versus a similar and more mature x86 platform, the AMD EPYC 7742, we improve the energy efficiency by 1.2× (55 vs 45 tok/s/mW) [3].

References

[1] Nick Brown and Maurice Jamieson. 2024. Performance characterisation of the 64-core SG2042 RISC-V CPU for HPC. arXiv:2406.12394 [cs.DC]
[2] Jiazhi Jiang et al. 2023. Characterizing and Optimizing Transformer Inference on ARM Many-core Processor. In *ICPP '22* (Bordeaux, France). Article 20, 11 pages.
[3] Tommaso Pegolotti et al. 2023. QIGen: Generating Efficient Kernels for Quantized Inference on Large Language Models. arXiv:2307.03738 [cs.LG]
[4] Zhihang Yuan et al. 2024. LLM Inference Unveiled: Survey and Roofline Model Insights. arXiv:2402.16363 [cs.CL] https://arxiv.org/abs/2402.16363
[5] Xiao et al. Fu. 2024. Optimizing Attention by Exploiting Data Reuse on ARM Multi-core CPUs. In *ICS '24* (Kyoto, Japan). 137–149.
[6] Georgi Gerganov. 2025. llama.cpp. https://github.com/ggerganov/llama.cpp.
[7] Chiyo Wang. 2024. PerfXLM: A LLM Inference Engine on RISC-V CPUs. In *RISC-V Summit Europe*. Presented at the RISC-V Summit Europe 2024.