V-Seek: Accelerating LLM Reasoning on Open-hardware Server-class RISC-V Platforms

Javier J. Poveda Rodrigo[†], Mohamed Amine Ahmdi[†*], Alessio Burrello[‡], Daniele Jahier Pagliari[‡] and Luca Benini[**]

[†]DAUIN, Politecnico of Turin, Turin, Italy
*Integrated Systems Laboratory (IIS), ETH Zurich

Abstract

The recent exponential growth of Large Language Models (LLMs) has relied on GPU-based systems. However, CPUs are emerging as a flexible and lower-cost alternative, especially when targeting inference and reasoning workloads. RISC-V is rapidly gaining traction in this area, given its open and vendor-neutral ISA. However, the RISC-V hardware for LLM workloads and the corresponding software ecosystem are not fully mature and streamlined, given the requirement of domain-specific tuning. This paper aims at filling this gap, focusing on optimizing LLM inference on the Sophon SG2042, the first commercially available many-core RISC-V CPU with vector processing capabilities. On two recent state-of-the-art LLMs optimized for reasoning, DeepSeek R1 Distill Llama 8B and DeepSeek R1 Distill QWEN 14B, we achieve 4.32/2.29 token/s for token generation and 6.54/3.68 token/s for prompt processing, with a speed up of up 2.9×/3.0× compared to our baseline.

Introduction

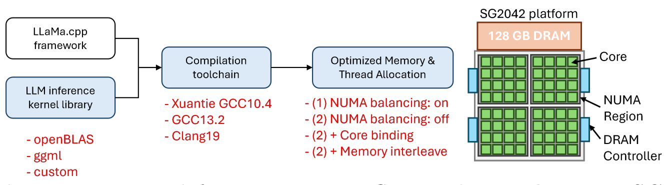
Hyperscalers (e.g., AWS) and AI deployment companies (e.g., OpenAI) typically accelerate LLM workloads using GPU clusters or dedicated accelerators such as Tensor Processing Units (TPUs). However, many-core CPU acceleration of LLMs has also been recently explored [1], as it provides advantages of lower hardware cost and enhanced flexibility, especially relevant for on-premise and low-latency edge servers. While existing studies mainly target x86 and ARM, recent many-core chips based on the flexible and open-source RISC-V Instruction Set Architecture (ISA) are relatively unexplored [2]. To bridge this gap, this work adapts and optimizes a state-of-the-art LLM inference framework (`llama.cpp` [3]) for the first commodity general-purpose, many-core RISC-V platform (Sophon SG2042 [2]).On two recent open-source LLMs optimized for reasoning (DeepSeek R1 Distill Llama 8B/QWEN 14B), we show speedups over a baseline `llama.cpp` implementation of up to 3.0× in token generation and 2.8× in prompt processing (i.e., prefill) at 4-bit precision, reaching a throughput of 4.32/2.29 and 6.54/3.68 tok/s, respectively. On vanilla Llama 7B, we achieve 6.63 and 13.07 tok/s for generation and prefill, i.e., a 4.3×/5.5× speedup w.r.t. the baseline, and 1.65× better w.r.t. the best-reported results on the SG2042 [4], while being competitive with CPU-based inference on the incumbent x86 architecture.

Methods

To explore the available alternatives to optimize LLM inference on RISC-V server-class platforms, we target the MILK-V Pioneer, comprising the 64-core Sophon SG2042 and 128GB of DRAM memory. A block diagram is shown in Fig. 1-center. We identify three directions from which the problem can be attacked, in SW, inspired by works on other architectures [5, 6, 7]:
i) Developing optimized and, if supported, quantized **kernels** for key LLM layers, that fully exploit the HW, coping with its memory infrastructure, pipeline, and vectorization; Fig.1-right, shows the pseudocode of our proposed kernel: first, the `fp32` input (vector or thin matrix) is quantized to `int8`; then, two nested loops are executed to perform a GEMV operation, the outermost iterating on the rows of input matrix A, and the innermost on its columns. After the column loop ends, de-quantization is applied, combining scale factors from A blocks and B to produce an output `fp32` value. This new kernel exploits the platform's vector units while also optimizing data locality.
ii) Choosing a suitable **compilation** toolchain, supporting advanced optimization passes and exploiting the available ISA extensions. In our case, kernels are compiled with the Xuantie fork of GCC 10.4, as it is the only one supporting the HW vector units of the Sophon SG2042. Instead, for the whole `llama.cpp` framework, we consider two alternatives: GCC 13.2, and Clang 19 (Xuantie GCC 10.4 is not compatible with the latest `llama.cpp` release).
iii) Optimizing model **mapping**, specifically pages/thread allocation, addressing this type of system's complex memory hierarchy. Namely, we optimize Non-uniform Memory Access (NUMA) latency exploring different *numactl* options combined in 4 policies: i) NUMA Balancing on, all other options off, ii) all options off, iii) Balancing off+Core Binding on, iv) Balancing off+Memory Interleaving on.

We apply our optimizatons to the `llama.cpp` [3] framework, testing on 3 open-source LLMs of increas-

*Corresponding author: `javier.poveda@polito.it`



Pseudocode: C = A x B
```
$B_{int8}$ = fp32_to_int8($B_{fp32}$)
(for col in columns($B_{int8}$))
for r in rows($A_{int4}$) with step 2
  for c in columns($A_{int4}$) with step 2x32
    $A_{int8}$[r,c:c+15],$A_{int8}$[r,c+16:c+31] = int4_to_int8_split($A_{int4}$[r,c:c+31])
    ... x4
    $A_{int8}$[r+1,c+32:c+47],$A_{int8}$[r+1,c+48:c+63] = int4_to_int8_split($A_{int4}$[r+1,c+32:c+63])
    $C_{int8}$[r] += vectorized_dot_product($A_{int8}$[r,c:c+15], $B_{int8}$[c:c+15])
    ... x8
    $C_{int8}$[r+1] += vectorized_dot_product($A_{int8}$[r+1,c+48:c+63], $B_{int8}$[c+48:c+63])
  $C_{fp32}$[r] = de-quantize($C_{int8}$[r]); $C_{fp32}$[r+1] = de-quantize($C_{int8}$[r+1])
```

Figure 1: *From left: optimization flow and contributions. SG2042 block diagram. Pseudocode of the proposed kernel.*

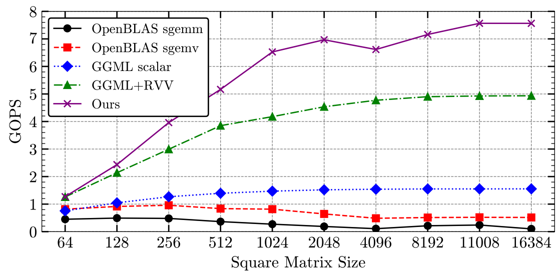


Figure 2: *Matrix vector multiplication size scalability test*

ing size, with Q4_0 quantization (vanilla Llama 7B, DeepSeek R1 Distill Llama 8B, DeepSeek R1 Distill QWEN 14B, referred to as 7B, 8B and 14B below).

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 test-generated tokens. **Kernel Scaling.** Fig.2 shows the single-thread scalability of multiple baseline kernels (llama.cpp's GGML and OpenBLAS's defaults) and of our proposed one. Compared to the best baseline, we improve the GOPS by +38.3% on average, peaking at +56.3% at matrix size 1024.

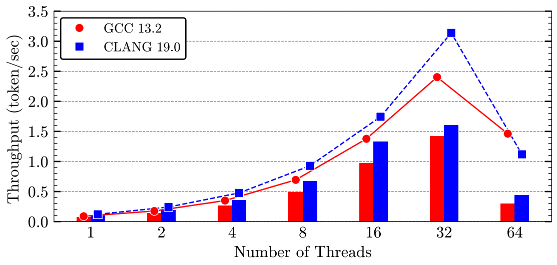


Figure 3: *Compilers comparison scaling the n. of threads for DeepSeek's 8B model token gen., Bar, and prefill, Line.*

Compiler exploration. In Fig.3, we evaluate DeepSeek's 8B model inference when compiling with Clang or GCC, using our proposed kernel. Clang 19 constantly outperforms GCC 13.2, reaching average performance gains 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 behavior is attributed to the default NUMA balancing policy, which is suboptimal for LLM inference due to the predictable nature of the workload, leading to a high number of thread and memory page migrations.

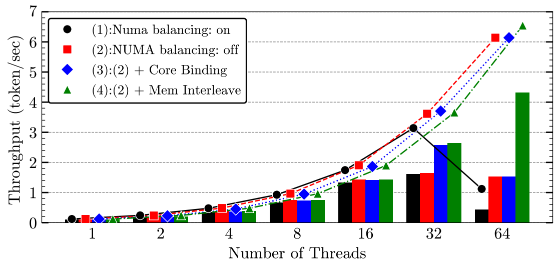


Figure 4: *NUMA policies exploration on DeepSeek's 8B model. Token generation shown with bars, prefill with lines.*

NUMA policy impact. Indeed, with the NUMA balancing off and memory interleaving on, as expected, we achieve the best results for both token generation (4.32 tokens/s) and for prefill (6.54 tokens/s) with 64 threads, thanks to the strong reduction in memory page migration.

Overall, thanks to our optimizations, the 7B, 8B and 14B LLMs reach a maximum throughput of 13.07/6.54/3.68 tok/s respectively, outperforming a baseline `llama.cpp` by up to 5.5×/2.9×/3×. Compared to the best reported result on the SG2042 [4], we improve the peak throughput on LLama 7B by 1.65×. Versus a similar and more mature x86 platform, the 64-cores AMD EPYC 7742, we improve the energy efficiency by 1.2×(55 token/s/mW vs 45 token/s/mW) [8].

References

[1] Haihao Shen et al. *Efficient LLM Inference on CPUs.* 2023. arXiv: `2311.00502 [cs.LG]`.

[2] Nick Brown and Maurice Jamieson. *Performance characterisation of the 64-core SG2042 RISC-V CPU for HPC.* 2024. arXiv: `2406.12394 [cs.DC]`.

[3] Georgi Gerganov. *llama.cpp.* `https://github.com/ggerganov/llama.cpp`. Accessed: 2025-01-21. 2025.

[4] Chiyo Wang. "PerfXLM: A LLM Inference Engine on RISC-V CPUs". In: *RISC-V Summit Europe.* Presented at the RISC-V Summit Europe 2024. 2024.

[5] Zhihang Yuan et al. *LLM Inference Unveiled: Survey and Roofline Model Insights.* 2024. arXiv: `2402.16363 [cs.CL]`. URL: `https://arxiv.org/abs/2402.16363`.

[6] Xiao et al. Fu. "Optimizing Attention by Exploiting Data Reuse on ARM Multi-core CPUs". In: *ICS '24.* Kyoto, Japan, 2024, pp. 137–149. ISBN: 9798400706103.

[7] Jiazhi Jiang et al. "Characterizing and Optimizing Transformer Inference on ARM Many-core Processor". In: *ICPP '22.* Bordeaux, France, 2023. ISBN: 9781450397339.

[8] Tommaso Pegolotti et al. *QIGen: Generating Efficient Kernels for Quantized Inference on Large Language Models.* 2023. arXiv: `2307.03738 [cs.LG]`.