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

Javier J. Poveda Rodrigo, Mohamed Amine Hamdi, Cyril Koenig, Alessio Burrello, Daniele Jahier Pagliari, Luca Benini.
Corresponding author: javier.poveda@polito.it

Politecnico di Torino

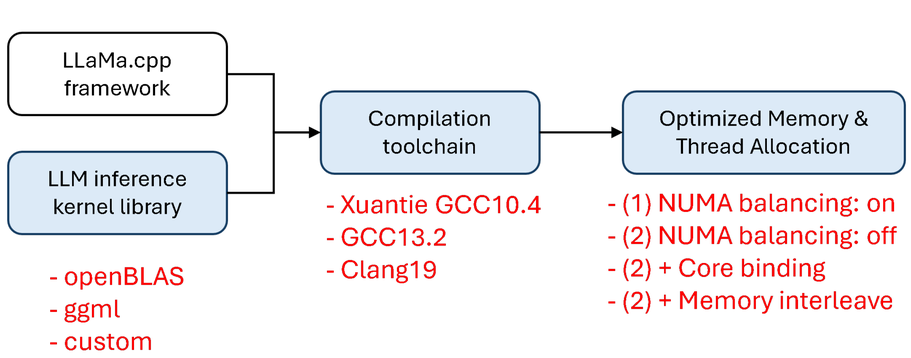
ETH zürich

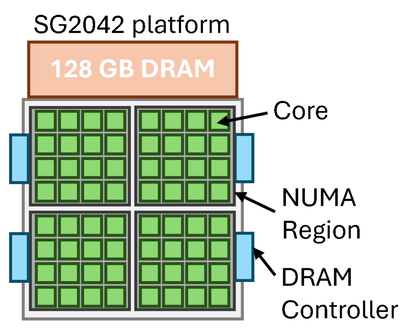
1. Introduction and Background

While nowadays GPUs are the computational core of LLM inference, this paper adapts and optimizes a state-of-the-art LLM inference framework, `llama.cpp`, for the first commodity general-purpose, many-core RISC-V platform, the Sophon SG2042. **Takeaways**:

- **Hardware-specific optimizations across the SW inference stack:** Hand-tuned kernels, NUMA Policy adaptation, optimal compilation toolchain.
- **Target Models:** Cutting-edge LLM Reasoning Models. DeepSeek R1 Distill Llama 8B and QWEN 14B.
- **Performance:** we achieve 3.0× speedup compared to the unoptimized llama.cpp baseline in token generation and 2.8× speedup in prompt processing at 4-bit precision, corresponding to throughputs of 4.32/2.29 tok/s and 6.54/3.68 tok/s, respectively. Our results improve by 1.65× the previously best-reported performance on the SG2042.

2. V-Seek Optimization Strategies



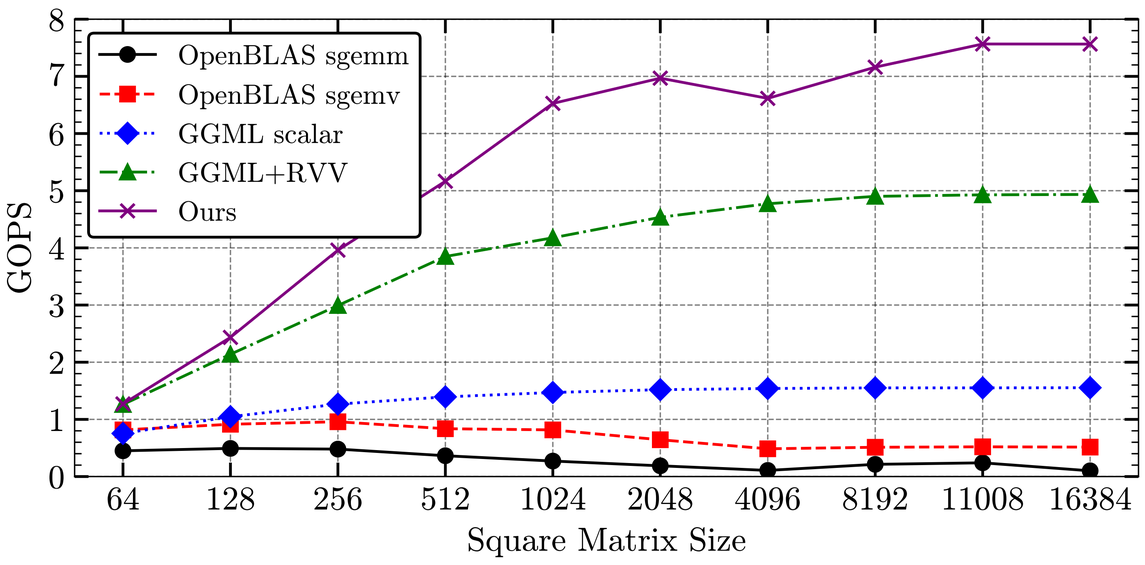


Pseudocode: C = A x B

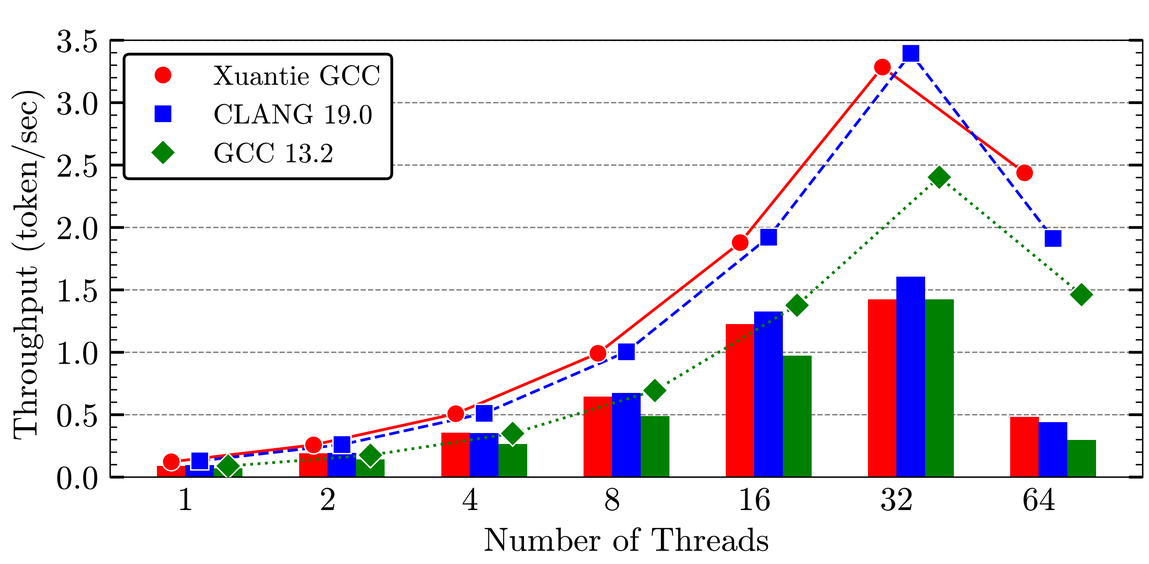
```
B_int8 = fp32_to_int8(B_fp32)
(for col in columns(B_int8))
for r in rows(A_int4) with step 2
  for c in columns(A_int4) with step 2x32
    A_int8[r,c:c+15],A_int8[r,c+16:c+31] = int4_to_int8_split(A_int4[r,c:c+31])
    ... x4
    A_int8[r+1,c+32:c+47],A_int8[r+1,c+48:c+63] = int4_to_int8_split(A_int4[r+1,c+32:c+63])
    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])
```

Left-to-right: optimization flow and contributions. SG2042 block diagram. Pseudocode of the proposed kernel.
We have identified three primary software-driven approaches for addressing optimization to exploit the SG2042 platform:
(i) Developing **optimized**, and where possible quantized, **kernels** for key LLM layers (GEMV and GEMM);
(ii) Selecting an appropriate **compilation toolchain**; we evaluate two alternative toolchains: Xuantie GCC 10.4, GCC 13.2 and Clang 19.
(iii) Optimizing the model **NUMA mapping**. We explore four distinct policies: (a) NUMA Balancing, (b) all options disabled, (c) Core Binding enabled, and (d) Memory Interleaving enabled.
Evaluated across three open-source LLMs: **vanilla Llama 7B, DeepSeek R1 Distill Llama 8B, and DeepSeek R1 Distill QWEN 14B**.

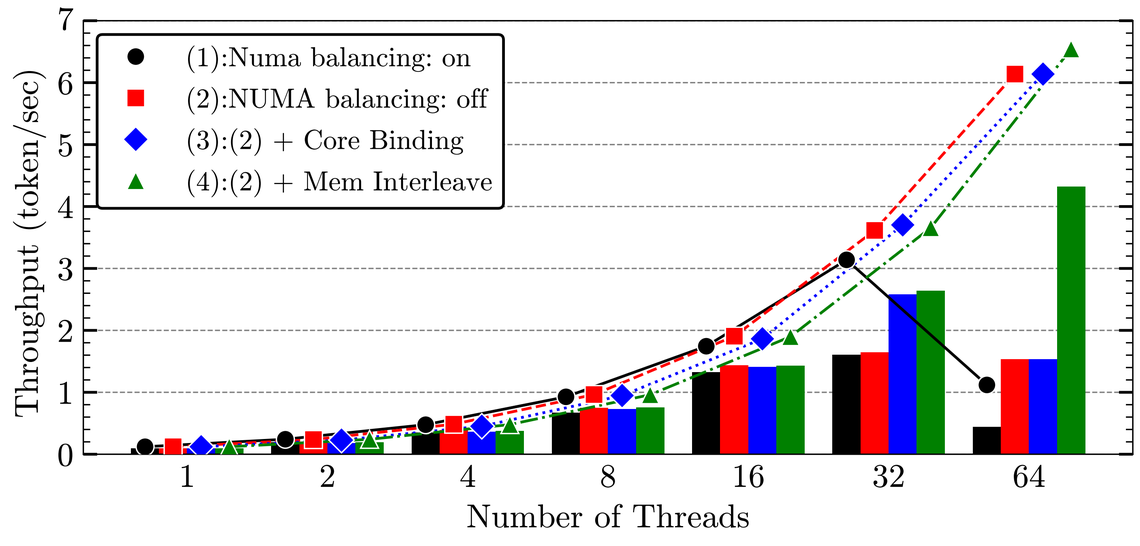
3. Results and State-of-the-art Comparison



Kernel Scaling. Compared to the best baseline (GGML + RVV extension), we improve the GOPS by +38.3% on average, peaking at +56.3% at matrix size 1024, thanks to fully exploiting vector units while improving memory re-usage.



Compilers. Clang 19 constantly outperforms GCC 13.2, reaching average performance gains of 34% and 25% for token generation (bar) and prefill (line), respectively. It is capable of producing maximum performance, outperforming Xuantie GCC.



NUMA Policies. With the NUMA balancing off and memory interleaving on, we achieve the best results for both token generation (4.32 tokens/s)(bar) and for prefill (6.54 tokens/s)(line) with 64 threads, thanks to the strong reduction in memory page migrations.

Source	Hardware	Inference Framework	Data format	Tok/s @2GHz	Efficiency (T/s/mW)
Baseline	SG2042	Pytorch + OpenBLAS	FP16	0.17	0.85
		Llama.cpp + OpenBLAS	FP16	0.55	5
			GGML Q4_0	1.55	13
PerfXLab	SG2042	PerfXLM	AWQ INT4	4.01	33
QIGen*	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.

SoA Comparisons. 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, 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).