Evaluating Memory Constraints of RISC-V Matrix Accelerators using gem5

Iago C. Aquino[1], **Casio P. Krebs**[1], **Lucas Wanner**[1], **Sandro Rigo**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Campinas, SP – Brazil

{i198921,c264953}@dac.unicamp.br, {lucas,sandro}@ic.unicamp.br

Abstract. *Matrix multiplication is a core operation in artificial intelligence workloads, often limited by memory bandwidth in modern computing accelerators. This study explores the architectural integration of a prototype RISC-V matrix extension using the gem5 simulator by modeling various memory hierarchy configurations, ranging from private caches to direct DRAM connections. Results demonstrate that strategic memory hierarchy placement significantly enhances computational throughput and efficiency. Our matrix implementation achieves 1.35x the performance of OpenBLAS using the same architectural state and 87% of the theoretical maximum.*

1. Introduction

The remarkable spread of artificial intelligence (AI) in contemporary computing drove a paradigm shift in processor and accelerator design, particularly due to the demand for manipulating extremely large datasets. The key operation for this workload is dense linear algebra, especially General Matrix-Matrix Multiplication (GEMM), which is the most used for training and inference in recent models.

To meet the demands of the ever-growing models, General-Purpose GPUs (GPGPUs) are employed, providing tens of billions of multiply-accumulate (MAC) operations per second, the base operation for GEMM. However, their usage is limited by its huge power requirements and memory restrictions. Addressing these bottlenecks, dedicated matrix acceleration units have been introduced to recent processors, with Intel's Advanced Matrix Extensions (AMX) [Kim et al. 2024] and ARM's Scalable Matrix Extension (SME) [Weidmann 2021], offering more efficient execution compared to traditional CPUs and GPGPUs, while also granting access to the larger memory pool of the system.

With its open-source and royalty-free model, the RISC-V [Waterman et al. 2014, RISC-V International 2025] instruction set architecture (ISA) has emerged as a compelling alternative for the data center space. Designed for flexibility, it has enabled the design of numerous products tailored to specific application domains, such as AI and high-performance computing (HPC). However, while vector extensions have already been standardized, matrix extensions remain under development, limiting GEMM performance and leading to proprietary extensions [Alibaba Cloud 2023, SiFive 2024] with fragmented software support.

Another key design challenge for next-generation matrix accelerators lies in balancing computational throughput with memory bandwidth. Without adequate data

delivery, even highly capable compute units risk remaining underutilized. However, mathematical analysis alone is insufficient to address this problem. A comprehensive understanding of how accelerators interact with different levels of the memory hierarchy is essential for guiding architectural optimization, making simulation necessary to achieve deeper insights.

The gem5 simulator [Binkert et al. 2011, Lowe-Power et al. 2020], a widely used, open-source, cycle-level simulation platform for computer architecture research, can provide enough information to guide accelerator projects. Its modular design enables full-system and microarchitectural simulation of modern CPU, memory, and peripheral components across multiple ISAs, including RISC-V. Since it can simulate pipeline stages, memory subsystems, and custom instructions, it can be employed for early-stage architectural exploration, especially when RTL is unavailable or too costly to modify.

In this work, we extend the gem5 simulator to model a prototype RISC-V matrix extension and evaluate its integration at different levels of the memory hierarchy, aiming to identify how architectural placement impacts bandwidth, latency, and arithmetic intensity in matrix-heavy workloads. Our custom implementation achieves up to 1.35x the performance of OpenBLAS vector-based kernel and reaches 87% of the theoretical peak throughput.

The rest of the paper is organized as follows: Section 2 provides an overview of related work on GEMM workloads and accelerator simulation, Section 3 describes the methodology used to implement and evaluate the RISC-V matrix extension within the gem5 simulator. Section 4 presents the simulation setup and discusses the architectural parameters used for matrix and vector comparisons. Section 5 details the experimental results, including performance across memory hierarchies and comparison with OpenBLAS. Finally, Section 6 concludes the work and outlines directions for future research.

2. Related Work

Achieving the best performance of a given hardware is a question of balancing the computing power of the chip with the ability of the system to keep it fed with data. Bigger and bigger accelerators are being built for recent tasks, but a lot of thought is still necessary to sustain this performance. For artificial intelligence, GEMM is the predominant task, which is memory-bound, as shown by a variety of works proposing solutions for memory hierarchy limitations and custom accelerators, as is done by [Yessin et al. 2014] and [Wang et al. 2024].

Literature already goes over arithmetic and memory balance, with [McCalpin 1995] proposing a mathematical formula for it (Equation 1). Although he proposes the use of a sustained memory operations to mitigate variations, there are multiple inconsistencies in memory performance that need simulation to be accounted for, as is the case of DRAM row management or cache parallelism, which affects latency.

$$B_{\text{machine}} = \frac{\text{Peak FLOPs/cycle}}{\text{Sustained Memory Ops/cycle}}. \tag{1}$$

The work of Goto [Goto and Geijn 2008], a reference in the implementation of linear algebra operations for CPUs, does an extensive job on optimizing the GEMM operation

for multiple levels of cache, as is the case in modern systems. This strategy mitigates some of the memory inconsistencies, but cannot be directly applied to our implementation, because we want to evaluate the requirements of a matrix accelerator fetching data at different memory levels, in other words, our accelerator does not benefit directly from the partitioning of data proposed by Goto. For some configurations, we cannot assume three or even two levels of cache. Furthermore, our accelerator design differs from the CPUs targeted by Goto in the communication between the scalar core and the functional units that are actually crunching numbers, which implies more data movement between both when a similar strategy is used.

Although this may seem like a disadvantage, the work done by [Volokitin et al. 2023] shows that RISC-V is a promising architecture for future HPC systems and that there is space for existing optimizations for ARM and x86 to be applied on RISC-V systems, the limitation now being the variety of available devices.

Several prior works have leveraged gem5 to evaluate accelerator integration. For instance, gem5-SALAM [Rogers et al. 2020] introduces a LLVM-based interface for accelerator modeling, while gem5-accel [Vieira et al. 2024] enables pre-RTL simulation of accelerators as memory-mapped devices. However, both approaches model accelerators as external devices and do not directly support custom ISA extensions or tight integration into the CPU instruction stream.

In contrast, our work extends gem5's RISC-V ISA with custom matrix instructions, matrix register file, and a dedicated datapath, enabling fine-grained simulation of a tightly-coupled matrix accelerator. By integrating it at different points in the memory hierarchy, from private L1 to DRAM, we capture bandwidth, latency, and performance counters, and characterize its behavior.

3. Methodology

This section describes the methodology adopted to design and evaluate the proposed RISC-V matrix extension prototype within the gem5 simulator. We outline the architectural choices made, the different accelerator configurations considered, the target GEMM workload used for testing, and the modifications applied to gem5.

3.1. RISC-V Matrix Extension

The standardization of matrix operations within the RISC-V ecosystem is currently being pursued through parallel efforts, but with slightly different approaches. One proposal aims to reuse the vector register file to represent matrices, adding instructions that handles vectors as linearized matrices without introducing new state. A second suggestion reuses the vector register file, but includes accumulator registers to increase the performance of MAC operations and reduce data movement during GEMM. The last option adds an entirely new set of matrix register state, operating independently of RVV.

RISC-V Vector was the first Single Instruction, Multiple Data (SIMD) to be consolidated to the ISA, being one of the requirements added to the RVA23 Profile, a base standard for future RISC-V compatible processors. This inclusion made RISC-V more suitable to HPC, AI and data center markets. It specifies a new set of registers with flexible size and SIMD-style vector operations for memory access, addition, multiplication and others. It stands out from ARM's SVE and Intel's AMX standards because of the

configuration options that allow for length agnostic code, allowing performance to scale while maintaining software compatibility.

The matrix extension proposals represent a step into specialized AI workloads, complementing the existing RVV with higher performance. The main distinction between both extensions lies in the data layout within registers. While RVV exposes one-dimensional vectors, most of the matrix extensions adopt a two-dimensional structure that enables more efficient matrix multiplication. Although promising, the proposals still lack a decision on topics like size configuration, supported data types, widening and interaction with existing vector instructions.

In this work, we implement a prototype matrix extension modeled after the RVV-independent option and inspired by one of the proposed RISC-V alternatives, defining 32 matrix registers (m0 - m31) organized as 4×4 tiles of 32-bit elements. With each matrix register having 512 bits, which matches the system cache line size, there is in total 2 KiB of additional architectural state. We introduce a set of custom instructions to load, store, zero, and perform fused multiply-accumulate (MAC) operations on these registers. Table 1 summarizes the instruction set implemented by our extension. All data movement between scalar, vector, and matrix registers occurs explicitly via memory operations, enabling fine-grained control and decoupling of matrix datapaths. Squared tiles were used to remove the need for register shape configuration, as this amount of flexibility was discarded by the task group, since it would increase the complexity of the matrix acceleration unit design.

Table 1. Description of instructions of the draft matrix extension

Instruction	Description
ml	Loads an entire matrix register from memory from a contiguous 512-bit block.
ms	Stores an entire matrix register to memory in a contiguous 512-bit block.
mls	Loads an entire matrix register from memory from 128-bit blocks separated by a stride value.
mss	Stores an entire matrix register to memory in 128-bit blocks separated by a stride value.
mzero	Copies the value zero to all positions of a register.
mmac	Computes the product of input registers and accumulates on the destination register.

3.2. System Architecture

When building an accelerator, providing sufficient data throughput is essential to avoid wasting computing cycles. However, it is not desirable to over-provide and waste area on resources that will sit idle. To balance these requirements, an accelerator can be attached to different levels of the memory hierarchy or even include a dedicated cache. Figure 1 illustrates some system architectures that were evaluated with simulation:

1a. Connected to a private L1 and then to a shared L2, the matrix accelerator has its own cache and can exchange data with the core without accessing DRAM.
1b. Connected to a private L1 and then to DRAM, similar to the previous, but forcing data to be exchanged through DRAM.

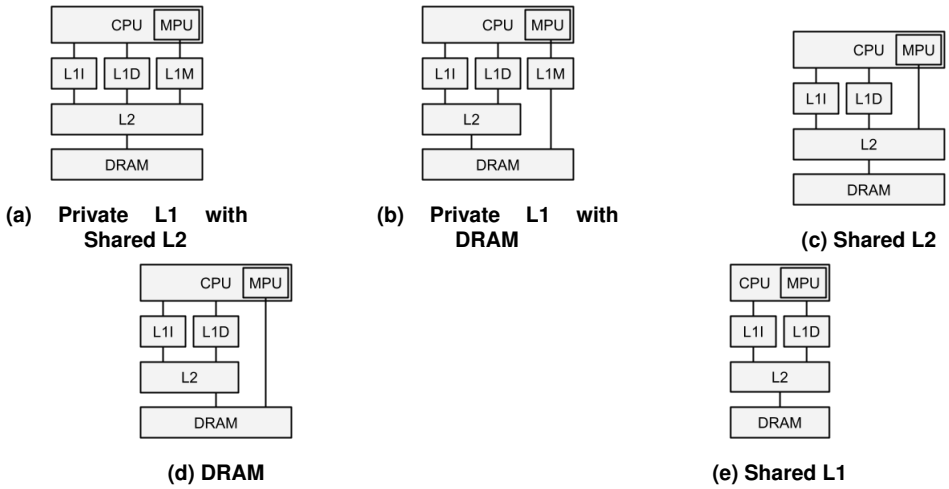

Figure 1. Different options for memory hierarchy.

1c. Connected to a shared L2, the matrix accelerator can exchange data with the core without accessing DRAM.
1d. Connected to DRAM, the matrix accelerator has to fetch all data from DRAM, introducing synchronization considerations, resembling a heterogeneous system.
1e. MPU sharing the same memory path as the scalar core.

3.3. Target Application

To evaluate our matrix extension and assess the performance of different memory hierarchy configurations, we focus on GEMM operations, known to be memory-bound.

We use OpenBLAS as a baseline, a well-optimized linear algebra library with support for RVV, serving as a representative of the current state-of-the-art GEMM implementation. The library was used without modification, ensuring a fair comparison with our custom kernel under identical architectural assumptions, including register size and number of functional units.

Inspired by Level 3 BLAS routines, we implemented a custom kernel targeting the proposed matrix instruction set optimized to maximize compute-to-memory ratio using blocked matrix multiplication, packing, multilevel memory optimization, and register reuse. It uses 24 matrix registers for output and the rest for input, structured as a 4x6 grid, enabling significant reuse of loaded data and minimizing stalls caused by memory accesses, as illustrated by Figure 2. The registers with a dashed outline correspond to the reuse that is possible once all the calculations are done, allowing the extension of the output panel. To ensure correctness, the results are validated against a baseline implementation of matrix multiplication in software.

Our benchmark consists of an application that receives the matrices' dimensions as a parameter and performs GEMM on randomly generated data for the given dimensions. The calls are instrumented with M5ops, special functions provided with gem5 that allow interaction between simulated system and host, to reset and dump gem5's statistics only on selected regions of the code, allowing us to eliminate the influence of the setup step. Using the same function signature, the three implementations can be swapped in the application, allowing us to compare all for results and performance.

Figure 2. Representation of register utilization by 4x6 kernel.

Exploring the concept of machine balance [McCalpin 1995], to achieve $B_{\text{machine}} = 1$, a system must feed the computational units at their peak rate without stalling for data. If the algorithm's *arithmetic intensity* (AI) is below this ratio, the kernel becomes *memory-bound*; if above, it is *compute-bound*.

A GEMM operation performs $2N^3$ floating-point operations (FLOPs) on $3N^2$ elements (two inputs and one output). The *theoretical arithmetic intensity* is therefore:

$$\text{AI}_{\text{instruction}} = \frac{\text{Floating-point Operations}}{\text{Loaded Elements}} = \frac{2N^3}{3N^2} = \frac{2}{3}N \quad [\text{FLOPs/element}] \tag{2}$$

This grows linearly with N, which explains why GEMM is considered a high-arithmetic-intensity kernel. However, the effective AI in hardware depends critically on how the data is fetched from memory, which in turn depends on the register blocking strategy, cache reuse, and sustained bandwidth.

At the microarchitectural level, the arithmetic intensity can be constrained by register size and instruction latency. Considering the parameters of our matrix extension, each register holds $\text{Elem}_{\text{reg}} = 16$ elements of size $\text{s}_{\text{elem}} = 32$ bits, and a MAC instruction does $\text{Ops}_{\text{elem}} = 8$ floating-point operations per element. If we consider issuing one MAC instruction every $\text{Lat}_{\text{mac}} = 4$ cycles, the peak per-cycle throughput is:

$$\text{Arithmetic Throughput}_{\text{peak}} = \frac{\text{Elem}_{\text{reg}} \times \text{Ops}_{\text{elem}}}{\text{Lat}_{\text{mac}}} = \frac{16 \times 4}{4} = 16 \text{ FLOPs/cycle}. \tag{3}$$

To achieve peak performance, the memory subsystem must supply enough data per cycle to sustain the *arithmetic throughput* (AT_{peak}), which, according to these parameters, translates to:

$$\text{Bandwidth}_{req} = \text{s}_{\text{elem}} \times \text{AT}_{\text{peak}} = 32 \times 16 = 512 \text{ bits/cycle} \tag{4}$$

Due to reuse, our kernel achieves a higher arithmetic intensity of:

$$\text{AI}_{\text{kernel}} = \frac{\text{Multiply Registers}}{\text{Load Registers}} \times \text{AT}_{\text{peak}} = \frac{24}{10} \times 16 = 38.4 \text{ FLOP/element}. \tag{5}$$

Finally, we compare the *effective Arithmetic Throughput* (6) obtained by simulation against the theoretical maximum predicted by (5) and (4).

$$\text{Arithmetic Throughput}_{\text{simulation}} = \frac{\text{Total FLOPs Executed}}{\text{Execution Cycles}} \quad [\text{FLOPs/cycle}] \tag{6}$$

3.4. gem5 Simulator

The gem5 simulator is a widely used, open-source, cycle-accurate simulation platform for computer architecture research. Its modular design enables full-system simulation with microarchitectural details of modern CPU, memory, and peripheral components across multiple ISAs, including RISC-V, ARM, and x86. Because gem5 can simulate pipeline stages, memory subsystems, and custom instructions, it is commonly employed for early-stage architectural exploration, especially when RTL is unavailable or too costly to modify. It has become an essential tool for both academia and industry due to its ability to simulate full-system architectures in detail and, unlike functional simulators as QEMU and Spike, it provides cycle-accurate results of the processor microarchitecture and cache systems. With more than 20 years of development, it provides a complete set of models for recent memory modules, devices, and ISAs, like RISC-V.

This framework was chosen for our evaluations due to our necessity to model the interactions between our matrix instructions and the memory hierarchy, as well as the CPU pipeline. We used gem5 to model different architectures where the matrix accelerator is attached at the L1 cache, L2 cache, or directly to main memory (DRAM). These scenarios reflect real architectural design choices, each with different latency and bandwidth characteristics. The simulator also provides fine-grained statistics, including cache hits and misses, memory queue occupancy, and port utilization, all of which are critical for understanding how effectively our matrix accelerator can be fed with data under varying conditions.

The first modification made to the RISC-V gem5 model was the addition of the new matrix instructions to its ISA Definition, following the same methodology as in [C. Aquino et al. 2024]. The numbered components in Figure 3 were modified.

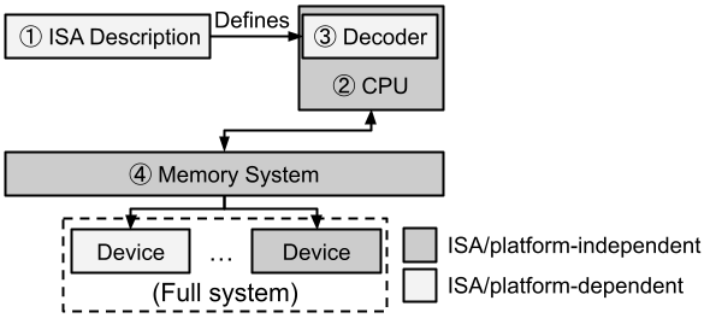

Figure 3. Simplified illustration of modified components from gem5 architecture.

1. ISA Description: defines the instruction formats, semantics, and how instructions are executed at the architectural level. This component was expanded with the matrix instructions present in our draft extension, as well as with the new registers and control registers.
2. CPU Model: takes care of the microarchitectural simulation, calling the instruction's behavior when it should execute. The O3CPU model was modified to include a new data port for matrix instructions.
3. Decoder: translates instruction bits (opcodes) into internal instruction representations that the CPU model can work with; this component is generated from the ISA Description.

4. Memory System: includes DRAM models and caches, was modified to accommodate the new data port.

4. Evaluation and Results

In this section, we present the evaluation of our proposed approach and discuss the results obtained from simulations. We explain the experimental setup, detail the workloads used, and compare the performance of matrix and vector implementations under different memory hierarchy configurations.

4.1. Simulation Setup

Our simulation environment is designed to isolate the impact of memory hierarchy and instruction set on computational throughput. Ubuntu 22.04 LTS was used with the GCC 13 version available at the time in the *riscv-gnu-toolchain* repository. To ensure a fair comparison between matrix and vector implementations, equivalent hardware parameters were set in gem5.

Both matrix and vector implementations operate on 512-bit registers, each accommodating 16 single-precision (32-bit) elements per register. Both have a single functional unit capable of performing MAC instructions, but each extension computes a different number of operations per instruction. Vector operations use an element-wise product (Figure 4a), computing 16 multiplications per instruction. On the other hand, matrix operations use an inner product between rows and columns (Figure 4b), computing 64 multiplications per instruction. To ensure equivalent peak compute throughput, vector MACs have one fourth of the latency of a matrix one, isolating performance differences to algorithmic and memory system behavior. One cycle for a floating point multiplication is unusual, but for the conditions of the experiment, it is an advantage for the vector model. Table 2 summarizes key architectural parameters simulated.

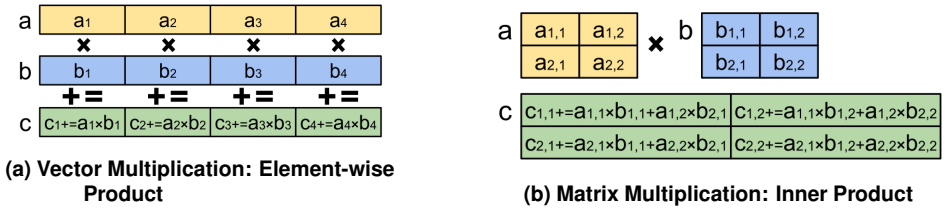

Figure 4. Multiplication strategy for vmacc.vv and mmac instructions.

Table 2. Parameters for vector and matrix operations

	Vector	Matrix
Operations per instruction	16	64
Number of Functional Units	1	1
Register Size (bits)	512	512
MAC Latency (cycles)	1	4
Peak Performance (flops/cycle)	16	16

4.2. Results

To define realistic workload sizes, we examined the convolution operations' dimensions reported by ConvBench [Alvarenga et al. 2024] datasets, which span from roughly 2 million to 60 million elements. To reduce the simulation effort while preserving equivalent computational demand, we chose square matrices with sizes 64, 128, 256, 384, 512, 768, 1024, 1280, 1536, 2048, 2560, 3072 and 4096, representing both small-scale inference and large transformer or deep neural network layers, and avoids the need to model every possible matrix shape. Every input size was simulated for each of the aforementioned memory hierarchies, generating a statistics file, which is used to capture the total simulated time and cycle count.

Figure 5 summarizes the effective performance of each scenario for matrix operations. In this graph, each curve is an exponential plot of the corresponding memory hierarchy, the vertical axis shows the effective performance in GMAC/s, while the horizontal axis shows the size of the matrices, on the bottom as the number of elements and on the top as the total size in MB of the matrices. Dashed lines indicate the sizes of the L1 and L2 caches, as well as the peak computing performance.

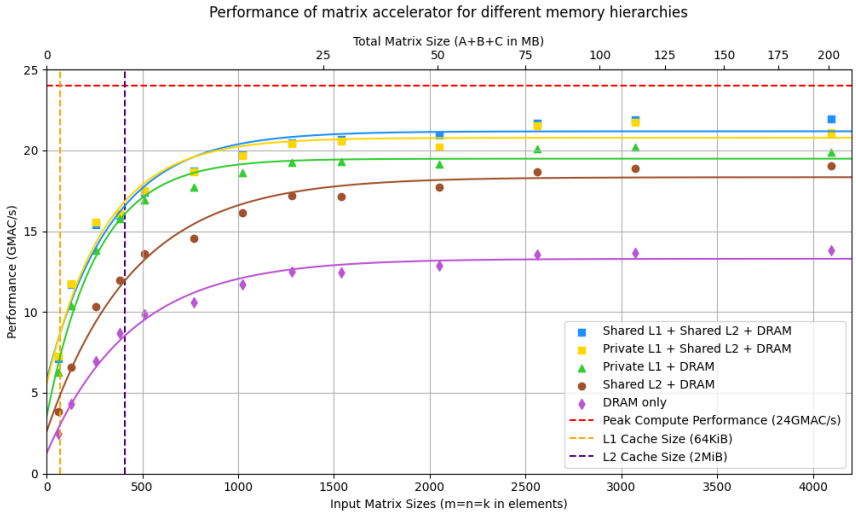

Figure 5. Comparison of accelerator performance for each memory hierarchy.

The first segment, between the y-axis and the orange dashed line, shows the smallest possible sizes, where every input matrix fits in the L1 cache and the performance is limited by the packing step. The second segment, between the the orange and indigo dashed lines, corresponds to the input matrices fitting in L2 cache. After the indigo dashed line, the performance of each kernel reaches its peak, especially at the points furthest to the right, allowing the characterization of its expected peak performance.

The results show that reusing the L1 cache between the accelerator and the scalar core (blue line) achieves the best performance, followed by the private L1 cache with shared L2 cache (yellow line), with less than 5% difference. As expected, connecting the accelerator directly to DRAM shows the lowest performance, which is aligned with the high bandwidth required by the GEMM algorithm, as shown by multiplying equation 4 by our simulated operating frequency of 1.5GHz resulting in a requirement of

$\text{Bandwidth}_{req} = 10$ GB/s. The bigger performance gap between DRAM and all cache scenarios reiterates the presence of data reuse.

The presence of L1 cache configurations at the top of the performance rankings highlights the algorithm's sensitivity to memory latency, while the configurations using L2 reveal that size and bandwidth are secondary to the performance, yet they alleviate the restrictions on algorithm tuning.

Figure 6 compares our private L1 matrix scenario, which corresponds to a more typical accelerator, against OpenBLAS. While vector instructions operate on one-dimensional data leading to a $\mathcal{O}(n^2)$ multiplications, matrix instructions leverage two-dimensional tile registers that enable multiple operations across rows and columns in a single instruction performing $\mathcal{O}(n^3)$ multiplications, effectively computing n times more operations while using the same number of loads. Our matrix-based implementation achieves 1.35 times better average performance for matrices of size 512 or higher compared to its vector counterpart, primarily due to its higher computational intensity and improved data reuse.

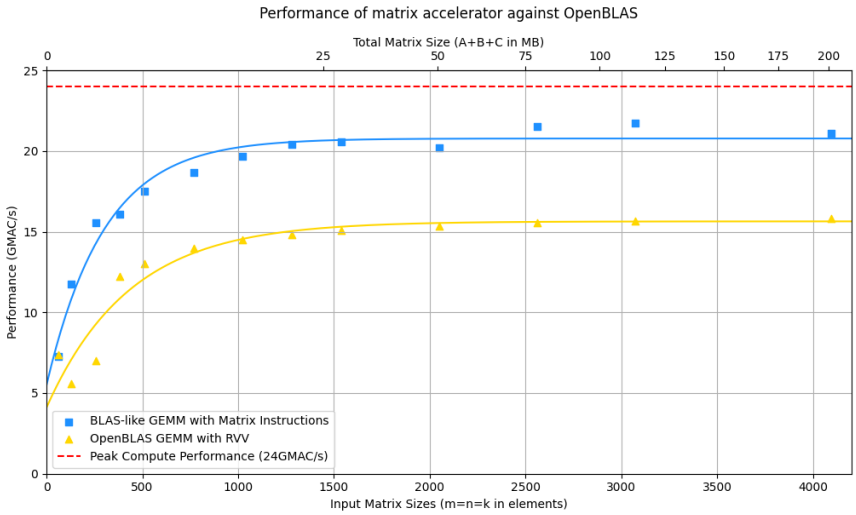

Figure 6. Comparison of Matrix and Vector implementations. Matrix GEMM achieves 1.35x RVV performance.

Finally, to better understand the performance limits of the evaluated configurations, we applied the Roofline model to create Figure 7, which relates achievable performance to both the peak compute capability of the system and the effective memory bandwidth. The horizontal red dashed line represents the peak compute performance of 48GFLOP/s, while the green dashed diagonal line corresponds to the DDR4 2400 MHz memory bandwidth limit of 19.2 GB/s.

The matrix implementation exhibits an operational intensity of approximately 4.8 FLOPs/Byte (blue dashed vertical line), appearing to the right of OpenBLAS (orange dashed vertical line) with 3.2 FLOPs/Byte, supporting the increased computational intensity of the matrix instructions. Also, the position of the measured matrix implementation rests closer to peak performance than OpenBLAS, 87% and 63% respectively, reinforcing its better utilization of available resources. The model shows that OpenBLAS's lower operational intensity puts it closer to a memory-bound regime

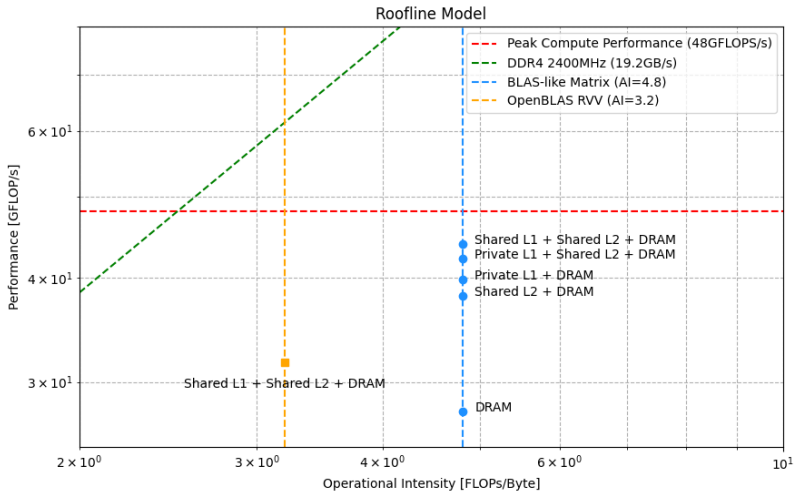

Figure 7. Roofline model comparing OpenBLAS and Matrix GEMM.

than our matrix implementation, which also could theoretically benefit from a higher peak compute, given that memory latency stays the same.

5. Conclusions and Future Work

In this paper, we presented an architectural exploration of matrix accelerators using a custom RISC-V matrix extension implemented within the gem5 simulator. Using GEMM workloads with varying matrix sizes and memory configurations, we demonstrated how data placement significantly impacts performance and bandwidth utilization.

Our results show that attaching the matrix accelerator closer to the processor core, particularly via shared L1 or L2 cache, improves arithmetic intensity and throughput by minimizing memory latency and maximizing reuse. The custom matrix kernel consistently outperformed the OpenBLAS-based vector baseline, highlighting the benefits of two-dimensional tile operations in both compute density and memory access efficiency.

This study confirms that gem5 is a powerful tool for modeling instruction-set-level accelerators with minimal changes, enabling detailed analysis of future ISA proposals such as RISC-V Matrix proposals. Our infrastructure and findings provide a starting point for architectural design and co-optimization of accelerators and memory systems in open hardware platforms.

In future research, we aim to develop and evaluate prototypes for other matrix proposals, including decoupled matrix accelerators and multi-core systems.

References

Alibaba Cloud (2023). XuanTie Matrix Multiply Extension Instructions. https://riscv.org/blog/2023/02/xuantie-matrix-multiply-extension-instructions/.

Alvarenga, L., Ferrari, V., Souza, R., Pereira, M., and Araujo, G. (2024). Convbench: A comprehensive benchmark for 2d convolution primitive evaluation.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N.,

Hill, M. D., and Wood, D. A. (2011). The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7.

C. Aquino, I., Wanner, L., and Rigo, S. (2024). *Architectural Simulation with gem5*, chapter 4, pages 92–118. Sociedade Brasileira de Computação, São Carlos, SP.

Goto, K. and Geijn, R. A. v. d. (2008). Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3).

Kim, H., Ye, G., Wang, N., Yazdanbakhsh, A., and Kim, N. S. (2024). Exploiting Intel Advanced Matrix Extensions (AMX) for Large Language Model Inference. *IEEE Computer Architecture Letters*, 23(1):117–120.

Lowe-Power, J., Akram, A., Amin, R., Hill, M. D., Wood, D. A., Chen, D. H., Hsu, L., Krishna, T., Agarwal, N., Wright, A. R., et al. (2020). The gem5 Simulator: Version 20.0+. arXiv preprint arXiv:2007.03152.

McCalpin, J. (1995). Memory bandwidth and machine balance in high performance computers. *IEEE Technical Committee on Computer Architecture Newsletter*, pages 19–25.

RISC-V International (2025). RISC-V International. https://riscv.org/.

Rogers, S., Slycord, J., Baharani, M., and Tabkhi, H. (2020). gem5-salam: A system architecture for llvm-based accelerator modeling. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 471–482.

SiFive (2024). Sifive proposal for risc-v ame extension. https://lists.riscv.org/g/tech-attached-matrix-extension/topic/sifive_proposal_for_risc_v/110189555.

Vieira, J., Roma, N., Falcao, G., and Tomás, P. (2024). gem5-accel: A pre-rtl simulation toolchain for accelerator architecture validation. *IEEE Computer Architecture Letters*, 23(1):1–4.

Volokitin, V., Kozinov, E., Kustikova, V., Liniov, A., and Meyerov, I. (2023). Case Study for Running Memory-Bound Kernels on RISC-V CPUs. In Malyshkin, V., editor, *Parallel Computing Technologies*, pages 51–65, Cham. Springer Nature Switzerland.

Wang, C., Song, P., Zhao, H., Zhang, F., Wang, J., and Zhang, L. (2024). High-Utilization GPGPU Design for Accelerating GEMM Workloads: An Incremental Approach. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5.

Waterman, A., Lee, Y., Patterson, D. A., and Asanovic, K. (2014). The RISC-V instruction set manual, volume I: User-level ISA, version 2.0. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54*, page 4.

Weidmann, M. (2021). Introducing the scalable matrix extension for the armv9-a architecture. https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/scalable-matrix-extension-armv9-a-architecture.

Yessin, G., Badawy, A. H. A., Narayana, V., Mayhew, D., and Ghazawi, T. E. (2014). "CERE": A CachE Recommendation Engine: Efficient Evolutionary Cache Hierarchy Design Space Exploration. In *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, pages 566–573.