RISC-V Meets RDBMS: An Experimental Study of Database Performance on an Open Instruction Set Architecture

Yizhe Zhang
University of New South Wales
Sydney, Australia
yizhe.zhang1@unsw.edu.au

Zhengyi Yang
University of New South Wales
Sydney, Australia
zhengyi.yang@unsw.edu.au

Bocheng Han
University of New South Wales
Sydney, Australia
bocheng.han@unsw.edu.au

Haoran Ning
Macquarie University
Sydney, Australia
haoran.ning@students.mq.edu.au

Xin Cao
University of New South Wales
Sydney, Australia
xin.cao@unsw.edu.au

John Shepherd
University of New South Wales
Sydney, Australia
j.a.shepherd@unsw.edu.au

Guanfeng Liu
Macquarie University
Sydney, Australia
guanfeng.liu@mq.edu.au

ABSTRACT

RISC-V, an open and extensible instruction set architecture, has gained significant attention in both academia and industry for its potential to reshape processor design. Among its numerous instruction set architecture (ISA) extensions, RISC-V Vector Extension (RVV) Version 1.0 introduces a novel approach to scalable and flexible vector computation, which holds particular promise for data-intensive workloads such as those found in modern database systems. In this paper, we systematically explore the implications of RISC-V architectural features, especially RVV 1.0, for database execution engines. We begin with an overview of the RISC-V ISA and its vector and scalar extensions relevant to data processing, then evaluate open source database systems compiled and executed on RISC-V platforms using industry-standard benchmarks such as TPC-H to measure performance and identify bottlenecks. Our experiments span various RISC-V ISA extensions, including the V, Zfh, Zknd, and compressed instruction sets, and assess their impact on execution speed, memory usage, and binary size. Our results demonstrate that mature existing database systems do not effectively leverage RISC-V ISA capabilities, with most extensions providing minimal performance improvements through change of database compilation parameters: the Vector Extension yields a performance improvement of less than 5%, while some extensions like Zknd can reduce performance by up to 32%. However, manually optimized RVV implementations achieve up to 10× speedups for specific query types, indicating substantial untapped potential. These findings reveal both the current limitations and future opportunities of RISC-V for database workloads, demonstrating that realizing the architecture's full potential requires hardware-aware optimization beyond current compiler capabilities and providing guidance for future system designs and hardware-software co-optimization.

VLDB Workshop Reference Format:
Yizhe Zhang, Zhengyi Yang, Bocheng Han, Haoran Ning, Xin Cao, John Shepherd, and Guanfeng Liu. RISC-V Meets RDBMS: An Experimental Study of Database Performance on an Open Instruction Set Architecture. VLDB 2025 Workshop: 16th International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures (ADMS25).

Zhengyi Yang is the corresponding author.

PVLDB Artifact Availability:
The code used in the case study "Manual RVV-Aware Query Optimization" is available at: https://github.com/mocusez/RVV-Parquet-Test

1 INTRODUCTION

RISC-V, an open and extensible instruction set architecture (ISA), has attracted substantial interest from both academia and industry due to its potential to fundamentally reshape modern processor design. Originally introduced in 2010 as a clean-slate ISA for research and industrial applications, RISC-V has evolved from a minimalist base integer ISA into a rich and modular architecture that includes a wide range of standardized extensions [32]. These include support for atomic operations (*A*), compressed instructions (*C*), single- and double-precision floating-point (*F/D*), half-precision arithmetic (*Zfh*), vector processing (*V*), virtual memory management (*Sv*), and various cryptographic primitives (*Zk**). The modular design allows hardware vendors to selectively incorporate extensions tailored to specific application domains, while the open governance model, led by RISC-V International, fosters broad collaboration and transparency across the ecosystem.

A major milestone was reached in 2024 with the commercial release of processors implementing the `RVA22` profile, which standardizes a suite of base and extension features for Linux-capable RISC-V systems. These processors also support the ratified Vector Extension Version 1.0 (`RVV 1.0`) [32], the first stable vector standard in the RISC-V ecosystem. Unlike traditional Single Instruction Multiple Data (SIMD) architectures found in other modern CPUs, `RVV 1.0` adopts a dynamic vector length model that enables binary portability between hardware with variable vector register widths. Processors conforming to the `RVA22` profile have achieved performance milestones as clock speeds exceeded 3 GHz and 14 nm process nodes [41]. Looking ahead, industry roadmaps suggest that

Proceedings of the VLDB Endowment. ISSN 2150-8097.

2025 may mark a pivotal moment for RISC-V, with the expected arrival of server-class processors designed for ultrascale data centers and high-performance computing (HPC) workloads [26].

Relational Database Management Systems (RDBMS) form the foundation of modern data management and have been extensively studied for their performance, robustness, and scalability. Although well-established in design, RDBMS are increasingly being re-examined in light of emerging hardware platforms and system-level primitives. Recent research has used heterogeneous compute accelerators (e.g. GPUs [20], FPGAs [15]), low-level networking frameworks (e.g. DPDK) and architectural innovations such as SIMD and Just-in-Time (JIT) compilation [21, 24] to accelerate query processing. These developments reflect a broader shift toward hardware-conscious database design, where tight integration with underlying architectural features can deliver significant performance benefits.

In this context, RISC-V has emerged as a promising open instruction set architecture (ISA) with growing hardware availability and ecosystem maturity. However, despite its rapid progress, the compatibility and performance of modern RDBMS on RISC-V remain relatively underexplored. Historically, practical deployment was hindered by limited compiler support and incomplete RISC-V integration in mainstream Linux distributions. These obstacles are now steadily being overcome, with toolchains reaching sufficient stability to support reliable experimentation. In particular, widely adopted Linux distributions, such as Debian and CentOS, have introduced increasingly stable RISC-V support [5, 31], and compiler toolchains for key system programming languages, including GCC, LLVM, Rust and Go, have made substantial strides toward production readiness [11, 18, 36].

These developments indicate that the RISC-V software ecosystem has become sufficiently robust to support systematic evaluation of database management systems on this architecture. In particular, ISA-level features such as vector processing (RVV), virtual memory, timestamp counters, and cryptographic extensions (e.g., `Zknd`) are expected to have measurable effects on the behavior of the database engine.

Building on this foundation, we present a practical investigation of the execution of database systems on RISC-V hardware, with a particular focus on evaluating the impact of ISA-level features on data analytic performance. Specifically, we conducted experiments using the TPC-H benchmark to assess how key architectural components, such as the Vector Extension (RVV), compressed instructions, floating-point support (Zfh and D), virtual memory, and cryptographic extensions (Zknd), influence query execution efficiency. This study provides concrete evidence of how design decisions at the ISA level affect the performance of real-world database on RISC-V platforms. Our key contributions are summarized as follows.

- We provide a review of the support for RDBMS in RISC-V, surveying a wide range of systems including PostgreSQL, MySQL, DuckDB, and commercial engines like Mimer SQL. Our study categorizes these systems based on their level of integration (e.g., official, community-maintained, patched) and identifies practical challenges such as dependency mismatches, compiler limitations, and missing architectural support that hinder smooth deployment on RISC-V platforms.
- We perform a detailed experimental analysis of several representative RISC-V ISA extensions: vector extension (`V`), floating-point support (`Zfh`, `D`), compressed instructions (`C`), and virtual memory support (`Sv`)—to evaluate their individual effects on OLAP query execution using TPC-H workloads. By isolating each extension, we quantify its performance impact and identify non-obvious trends, such as the limited effectiveness of compiler-driven vectorization and the minimal runtime influence of compressed instructions. To the best of our knowledge, previous work has not systematically evaluated these ISA components in the context of database query performance on RISC-V platforms.
- We demonstrate the performance potential of RVV 1.0 in accelerating relational workloads by manually optimizing vectorized query operators. Our hand-tuned implementations yield up to 10× speedups on scan, filter, and aggregation-heavy queries, highlighting the substantial performance gains achievable when explicit vectorization is employed in cases where compiler auto-vectorization falls short.

Paper Organization. The remainder of the paper is organized as follows. Section 2 provides a primer on the RISC-V features relevant to database execution. Section 3 surveys the support for RDBMS in RISC-V. Section 4 presents performance evaluations using TPC-H benchmarks.

2 RISC-V ARCHITECTURE PRIMER

To contextualize our study, we first outline RISC-V's core attributes and then examine its implications for database workloads through two lenses: the base ISA's modularity and the vector extension's potential for data process.

2.1 What is RISC-V?

RISC-V is an open standard instruction set architecture (ISA) designed with an emphasis on simplicity, modularity, and extensibility. Unlike proprietary ISAs such as x86 and ARM, RISC-V is publicly available and does not require licensing, enabling broad participation from academia, industry, and open-source communities. Its open nature supports various hardware implementations and facilitates research and development across a range of platforms—from embedded devices to high-performance computing systems. The ISA's modular structure also promotes hardware-software co-design, where system software and hardware architectures can be jointly optimized to meet domain-specific performance and functionality requirements such as database and AI [39].

2.2 Overview of RISC-V ISA

RISC-V is designed to be modular, extensible, and royalty-free, enabling academic research and industrial deployment across a wide range of platforms, from embedded microcontrollers to high-performance servers [3]. Its modular nature allows implementers to select instruction set extensions that meet their specific application requirements and to optionally define custom extensions when needed. This flexibility makes RISC-V particularly attractive for both cost-sensitive embedded systems and performance-critical domains.

The base ISA defines a minimal integer instruction set (RV32I/RV64I), which can be extended with standard extensions such as:

- **M**: Integer multiplication and division
- **A**: Atomic instructions for synchronization and concurrency
- **F/D**: Single and double-precision floating-point support
- **C**: Compressed 16-bit instructions for improved code density
- **V**: The vector extension for data-parallel workloads, critical for query processing and machine learning
- **Zicsr/Zifencei**: Control and status register access and instruction fence, respectively, both are essential for low-level system programming.

RISC-V's modularity also allows the inclusion of emerging extensions like `Zfinx`, which enables floating-point operations using integer registers, particularly useful for systems without dedicated floating-point units (FPUs). Other relevant extensions in database contexts include `Zba/Zbb` for bit manipulation and `Zk*` for cryptographic primitives.

For vectorized and parallel data processing, the `V` extension (RVV) introduces scalable vector registers and operations, which make RISC-V especially promising for accelerating scan, join, and aggregation operations in analytical workloads.

Its openness and growing ecosystem, including upstream support in compilers (e.g., GCC, LLVM), operating systems (e.g., Linux), and hardware simulators (e.g., QEMU), make RISC-V an increasingly attractive target for database system research and deployment.

We noticed that RISC-V ISA has proposals related to JIT and Transaction Memory [3], but unfortunately there is no progress in these areas.

2.3 Vector Extension (RVV 1.0)

The RISC-V Vector Extension Version 1.0 (RVV 1.0) represents a significant milestone as the first frozen vector instruction standard in the RISC-V ecosystem, establishing a stable ABI for vectorized computation. Unlike its provisional predecessors, RVV 1.0 introduces a production-ready, scalable vector ISA model that decouples the hardware vector length from the instruction semantics while maintaining binary compatibility. This design enables portable high performance execution of data-parallel workloads across heterogeneous hardware, which is a critical advantage for modern database engines [32].

Scalable Vector Lengths. RVV adopts a dynamic vector length model, where the maximum vector length (`VLEN`) is implementation-defined, and the active vector length (`VL`) is determined at runtime by `vsetvl` instruction. This allows a single binary to adapt to a wide variety of hardware configurations, from low-power edge devices to high-throughput server-class processors. For database systems with vectorized execution engines, this facilitates query plan portability without sacrificing hardware efficiency.

Vector Registers and Element Types. The architecture provides 32 vector registers (`v0–v31`), each capable of holding up to `VLEN` bits. These registers operate on configurable element types (standard element widths, `SEW`), which range from 8 to 64 bits for integers and floating point values. This flexibility is beneficial for processing columnar data formats (e.g., Arrow, Parquet), especially when dealing with compressed or dictionary-encoded columns that require type reinterpretation or width conversion.

Masking and Predicated Execution. RVV includes implicit predication using a dedicated vector mask register (`v0`). This supports masked operations that enable conditional execution on a per-element basis. For database queries, this allows selection predicates and null filtering to be vectorized efficiently without introducing control flow divergence, preserving SIMD lane utilization even under sparse filtering conditions.

Flexible Memory Access. The extension supports multiple memory access patterns, including unit-stride, strided, indexed, and segmented loads and stores. These are especially useful in scan and join operations where attribute data are often accessed in non-contiguous layouts due to projection or cache-optimized storage formats. Combined with the loop-stripmining behavior inherent in RVV, this enables efficient traversal over large datasets with minimal control overhead.

Theoretical Implications for Database Execution Engines. The vectorized execution paradigm of modern analytical databases aligns naturally with the capabilities of RVV. For example, tight loops over columnar buffers, used in filter, projection, and aggregation operators, can be directly mapped to RVV instructions, reducing interpretation overhead and improving throughput. Unlike fixed-length SIMD where compile-time vector width limits performance portability, RVV enables database engines to generate vectorized code that adapts to the available hardware, either via JIT or interpreter specialization. This makes RVV particularly suitable for execution engines targeting RISC-V backends in cloud or edge deployments.

3 DATABASE ECOSYSTEM ON RISC-V

Running relational database systems on RISC-V platforms introduces a multifaceted set of challenges, ranging from early-stage ecosystem maturity to architectural compatibility and ISA-specific performance tuning. In this section, we analyze the current state of the RISC-V database software stack, with a focus on toolchain availability, OS support, and compatibility of mainstream RDBMS engines.

3.1 Toolchain & OS Support

Before relational database systems can be effectively run on RISC-V platforms, a robust and well-developed software ecosystem is essential. One of the most critical milestones was the integration of RISC-V support into the mainline Linux kernel on 4.15, completed in 2017 [33]. This achievement, driven by the collaborative efforts of the open-source community, laid the foundation for operating system compatibility and hardware abstraction on RISC-V processors.

Building on this, major Linux distributions, including Debian [6], Fedora, Arch Linux, and Gentoo, have gradually incorporated RISC-V ports into their official repositories, enabling package management, system tooling, and essential libraries to function seamlessly on RISC-V systems. These distributions have not only provided base

Table 1: Milestones of RISC-V support across major toolchains

Year	Toolchain	Milestone
2017	GCC	RISC-V support introduced in GCC 7.2
2018	LLVM	Stable RISC-V backend in LLVM 9.0
2020	Rust	RISC-V promoted to Tier 2 support
2021	GDB	V extension debug supported in GDB 11.1
2023	Go	Official 64-bit RISC-V binary released
2024	GCC/LLVM	Full RVV1.0 support in both compilers

system compatibility, but have also actively maintained patches and build pipelines for RISC-V support.

Furthermore, the readiness of modern programming languages has played a key role in enabling the development of a database system on RISC-V. The Rust, Go, and C++ toolchains(GCC, GDB [4], LLVM) have all introduced stable support for RISC-V targets under Linux, as illustrated in Table 1, allowing developers to build and optimize systems-level software, including databases, without requiring platform-specific rewrites. The cross-language toolchain support simplifies the process of porting and optimizing database engines such as PostgreSQL, DuckDB, and TiDB.

Overall, the convergence of upstream Linux support, distribution-level packaging, and modern language toolchains has made it feasible to develop, compile, and deploy high-performance database systems on RISC-V, accelerating both research and production use of the architecture. ures like the vector extension or relaxed memory models.

3.2 RDBMS Compatibility

As shown in Table 2, the support for the RISC-V architecture varies significantly between the main relational database management systems (RDBMS). Among those with official support, **SQLite** demonstrates excellent portability, due to its clean C codebase and well-maintained engineering practices, allowing seamless compilation on RISC-V platforms. **PostgreSQL** has also officially merged architecture-specific patches, such as support for RISC-V spinlocks, into its mainline.

MySQL and MariaDB. For communities-maintained ports, the situation is more fragmented. **MySQL**, a widely used open-source RDBMS known for its plugable storage engine architecture and wide compatibility with legacy enterprise applications, has not officially adopted support for RISC-V. Although downstream distributions such as Debian and Gentoo maintain working patches —-notably those addressing access to CPU timestamp counters (e.g., `rdtime`) and atomic operations —- the upstream MySQL project has declined to merge these changes[23]. By contrast, **MariaDB**, a community-driven fork of MySQL that emphasizes extensibility and portability, has integrated equivalent RISC-V patches upstream. These include the architectural adjustments necessary for clean builds on RISC-V Linux [7]. MariaDB's more modular design, active collaboration with distribution maintainers, and broader platform testing have enabled smoother operation on RISC-V systems, positioning it as a more accessible option for open-source RDBMS experimentation on this architecture.

PostgreSQL. PostgreSQL is a widely adopted open-source relational database system known for its robustness, extensibility, and compliance with SQL standards. Designed with a strong focus on correctness and transactional integrity, PostgreSQL is commonly used in applications requiring complex queries, concurrency control, and advanced data types. Starting from version 16, official support for the RISC-V architecture has been incorporated into the upstream repository [29], allowing out-of-the-box compilation without the need for downstream patches. The PostgreSQL community has shown a proactive engagement in supporting emerging platforms, and RISC-V is no exception. Packages for RISC-V are actively maintained in major Linux distributions such as Debian and Arch Linux, ensuring accessibility and timely updates. This makes PostgreSQL one of the most mature and portable open-source databases available for experimentation and deployment on RISC-V systems.

SQLite, as a lightweight embeddable SQL database engine, maintains strong portability across platforms due to its minimalist design and careful engineering. Although the SQLite development team does not explicitly advertise RISC-V support, the system compiles and runs out-of-the-box on RISC-V without requiring any patches or architecture-specific modifications. This is largely attributed to its OS-level abstraction layer (VFS) and CPU-agnostic architecture, where platform independence is achieved through portable C code, leaving CPU-specific details to the compiler [13].

TiDB, while capable of running on RISC-V, has not yet integrated native support, citing the absence of continuous integration (CI) infrastructure for this architecture [37]. TiDB is a distributed SQL database designed for HTAP workloads, combining a stateless SQL layer (compatible with MySQL) with a distributed key-value storage engine inspired by Google Spanner and HBase [14]. Its architecture is implemented primarily in Go, which makes the maturity of the Go toolchain critical for successful compilation and execution. Although PerfXLab's 2023 experiment reported that TiDB could not be compiled or executed on RISC-V [28], our 2025 experiment demonstrates that TiDB can now be successfully built and run on RISC-V hardware. This progress is largely attributable to the ongoing stabilization and completeness of the Go language toolchain for the RISC-V Linux environment, which supports the TiDB's build system and runtime. The discrepancy between earlier and current findings underscores the rapid evolution of the RISC-V software ecosystem.

DuckDB. DuckDB is an in-process analytical database designed for OLAP-style workloads, featuring vectorized query execution, columnar storage, and seamless integration with data science ecosystems. Its lightweight architecture and zero-dependency deployment model make it well suited for embedded analytics and interactive environments such as notebooks [30]. In 2025, DuckDB has integrated CI testing for RISC-V using QEMU-based environments, demonstrating functional compatibility with RISC-V Linux [9]. However, the system depends on `jemalloc`—a high-performance memory allocator that improves multithreaded memory management efficiency—which in turn requires the `Zihintpause` extension. This optional RISC-V ISA feature enables more efficient spin-wait loops, which `jemalloc` leverages to reduce contention and

Table 2: RISC-V Support Status of Major RDBMS (as of 2025)

Database	Type	Open Source	Primary Language	Support	Patch	Distro Release	Binary Release
SQLite	OLTP	Yes	C	Official	No	Yes	No
Mimer SQL	OLTP	No	C/C++	Official	No	No	Yes
Oracle	OLTP	No	C/C++	No	N/A	No	No
SQL Server	OLTP	No	C/C++	No	N/A	No	No
PostgreSQL	OLTP	Yes	C	Official	No	Yes	No
MySQL	OLTP	Yes	C/C++	Community	Yes	Yes	No
MariaDB	OLTP	Yes	C/C++	Community	No	Yes	No
DuckDB	OLAP	Yes	C++	Community	No	No	No
ClickHouse	OLAP	Yes	C++	No	N/A	No	No
TiDB	HTAP	Yes	Go	Community	No	No	No
OceanBase	HTAP	Yes	C++	No	N/A	No	No

improve allocator performance under parallel workloads. Consequently, DuckDB runs reliably only on hardware or emulators that implement `Zihintpause`, limiting support on minimal RISC-V cores that omit this extension.

Mimer SQL. Mimer SQL is a lightweight relational database system primarily designed for embedded high-performance and real-time applications. It is known for its small footprint, support for standard SQL and the ability to operate in environments with stringent resource constraints, such as automotive and industrial control systems. Although it is proprietary software, Mimer SQL stands out as one of the few commercial database systems that officially supports RISC-V, offering pre-built binaries targeted at this architecture.

In contrast, several major systems—such as **Oracle**, **SQL Server**, **ClickHouse**, and **OceanBase**—have not disclosed any RISC-V support efforts to date. These systems are either tightly coupled to x86-specific optimizations or are closed-source, hindering community-driven porting.

4 PERFORMANCE EVALUATION

We perform a comprehensive performance evaluation of relational databases on RISC-V platforms, focusing on three key perspectives: (1) the overall impact of the base RISC-V ISA on analytical workloads (subsection 4.3), (2) the performance effects of enabling or disabling specific ISA extensions (subsection 4.4), and (3) the performance gains achievable through manual optimization using the RISC-V Vector Extension (RVV) (subsection 4.6). Before presenting the results, we first outline the experimental setup used in our study.

4.1 Test Environment

Hardware Setup. All native RISC-V experiments were conducted on the Milk-V Jupiter development board, equipped with a Spacemit M1 processor clocked at 1.8 GHz and paired with 16 GB LPDDR4X memory. The processor features an octa-core X60™ implementation (RV64GCVB) that compiles with RVA22 ABI and supports the RISC-V Vector Extension 1.0 (RVV 1.0) [22]. Passive thermal dissipation is provided via a copper heatsink. Persistent storage is handled by a FanXiang S500Pro NVMe SSD.

Software Stack. The operating system used is Bianbu Linux, a Debian-derived distribution customized by Spacemit, running the Linux kernel version 6.6. The database systems evaluated include PostgreSQL (v16.2), DuckDB (v1.3.0), and SQLite (v3.45.1, commit `e876e51a0`). PostgreSQL and SQLite were installed from the official Bianbu package repository. Compilation was performed using `GCC 14`, with `LLVM 20` used for comparison in specific experiments.

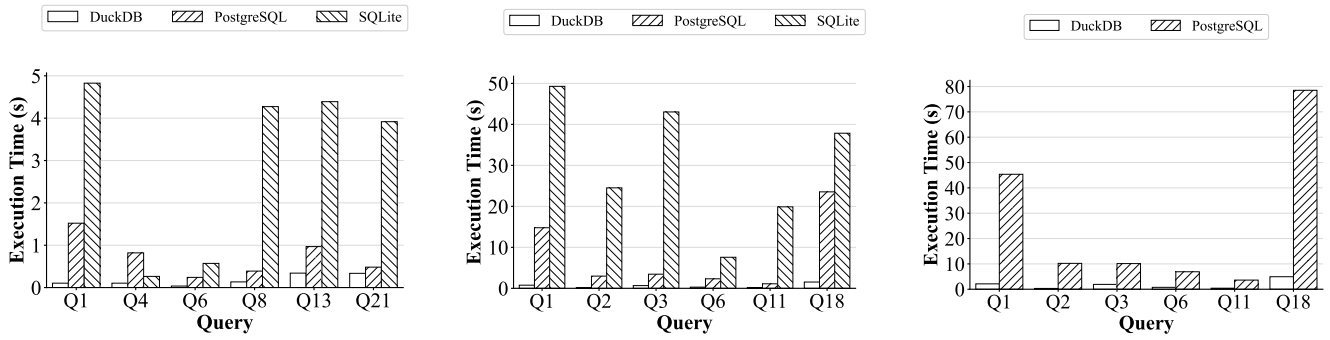
Emulated Environment. Some experiments, particularly those that involve instruction-level tuning or compiler comparisons not feasible on the hardware board(Virtual Memory Extension and `Zknd` extension), were conducted using QEMU 10. The host system featured an Intel® Xeon® Gold 6342 CPU at 2.80 GHz. The QEMU target was configured for RV64 with 8 emulated cores and 16 GB allocated memory. ISA extensions were selectively enabled depending on the specific experiment, allowing detailed control over architectural configurations.

Benchmark Configuration. We use the TPC-H benchmark [38], a widely adopted OLAP workload that simulates decision support queries on large datasets with minimal prior knowledge of query patterns. It consists of eight tables in the third normal form (3NF). Unless otherwise noted, experiments are conducted using a scale factor of 10 and all benchmark results report the mean of three independent runs.

4.2 How Does the RISC-V ISA Influence TPC-H Performance?

To understand the architectural impact of RISC-V on analytical query workloads, we examine how key ISA-level features influence performance when executing the TPC-H benchmark on emerging RISC-V platforms.

Can vector instructions accelerate core query operations? Yes. RISC-V's Vector Extension (RVV) Version 1.0, particularly with hardware supporting at least `Zvl256b` (256-bit vector register length), enables efficient data-level parallelism. This is beneficial for operations such as scan, filter, projection, and aggregation, all of which are fundamental to analytical queries.



(a) TPC-H (SF0.1) (b) TPC-H (SF1) (c) TPC-H (SF3)

Figure 1: Execution time comparison of DuckDB, PostgreSQL, and SQLite across scale factors.

What is the role of floating-point extensions in analytical workloads? Floating-point capabilities, such as `Zfh` (half-precision) and `D` (double-precision), are essential for numerical processing. Aggregation and statistical functions in TPC-H (e.g., `SUM`, `AVG`) rely on precise and efficient arithmetic operations, which these extensions help accelerate.

Can Compressed Instructions effect Execution Efficiency? The `C` extension (compressed instruction set) contributes to performance indirectly by reducing code size and alleviating instruction cache pressure. This is particularly beneficial for vectorized query engines, where complex, instruction-dense execution plans can suffer from instruction cache thrashing. By improving binary compactness, the C extension improves cache residency, which in turn helps sustain pipeline throughput under heavy workloads.

Is virtual memory support relevant for database workloads? Yes. Support for virtual memory, including standard paging and address translation mechanisms, is vital for running complex database systems under Linux. Activating features such as memory protection, process isolation, and fine-grained memory management, all of which are fundamental to stability and performance.

What about security and cryptographic extensions? While not directly affecting the speed of the query, cryptographic extensions such as `Zknd` can enhance data security by accelerating encryption and integrity checks. This makes them valuable in secure database environments, especially in cloud or multi-tenant deployments.

How important is the software stack? A well-developed software environment is critical, including RVV-aware compiler toolchains (e.g., LLVM with vector intrinsics), vector-friendly memory layouts, and an in-memory execution model—is critical. These components ensure that the hardware features are actually utilized in query execution, enabling meaningful performance gains.

Together, these features define a practical baseline for evaluating RISC-V in high-performance relational database systems. By focusing on microarchitectural characteristics, we can isolate the ISA's impact on analytical query workloads such as those represented by TPC-H.

4.3 Baseline TPC-H Benchmark Results

As shown in Figure 1a,Figure 1b and Figure 1c, DuckDB maintains consistent performance in all scale factors (SF0.1, SF1, SF3), successfully executing all queries within the 20 second threshold. PostgreSQL shows scalability challenges: while handling SF0.1 adequately, it exceeds the timeout on Query 17 and 20 at SF1, and these issues intensify at SF3 (Figure 1b,Figure 1c). SQLite demonstrates severe performance degradation with scale growth - though functional for Queries 6, 10–12, 14,16 at SF0.1 (Figure 1a), it fails most queries (including 17,20,22) at SF1 and becomes impractical at SF3. DuckDB is the only database that can run TPC-H SF10 in less than 5 minutes.

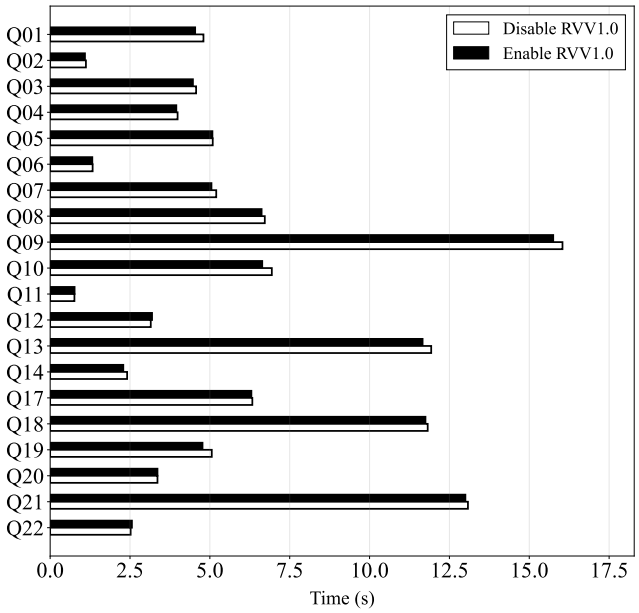
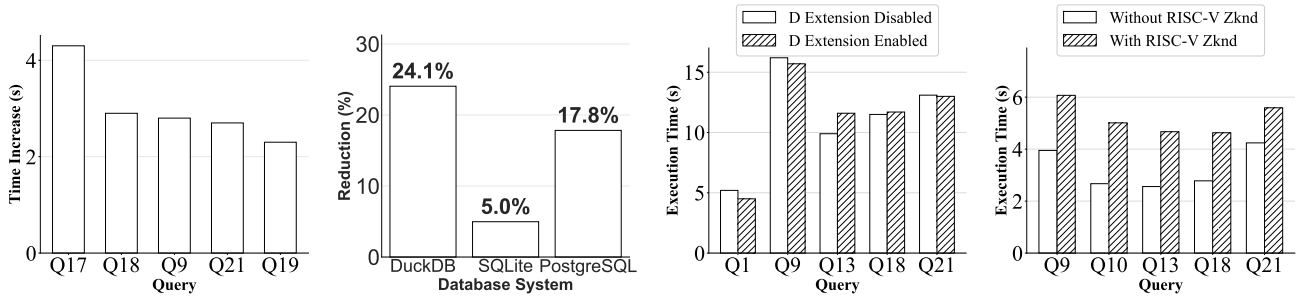


Figure 2: Comparison of total elapsed time on DuckDB for each Query with and without RVV1.0 on TPC-H SF10



(a) `Zfh` extension impact (SF10) **(b) `C` extension binary size** **(c) `D` extension impact (SF10)** **(d) `Zknd` extension impact (SF3)**

Figure 3: Performance impact of key RISC-V ISA extensions across database benchmarks.

4.4 RISC-V ISA Extension Impact Analysis

Given DuckDB's superior performance across all TPC-H scale factors in our baseline evaluation (subsection 4.3), we selected it as the primary subject to analyze the impact of RISC-V ISA extensions. As the only engine capable of completing the TPC-H SF10 benchmark in five minutes, DuckDB provides a stable and high-performing baseline, making it well suited for assessing how low-level architectural features affect execution efficiency.

To this end, we compile DuckDB under varying compiler configurations and with selected RISC-V extensions enabled, evaluating the effects on execution time, binary size, and responsiveness to vectorized and compressed data formats.

V Extension. The Spacemit M1 processor adopts the RVV 1.0 standard for the RISC-V vector extension. To evaluate its impact on database performance, we modified DuckDB compilation parameters to enable or disable RVV 1.0 support and executed the TPC-H benchmark at scale factor 10 (SF10). The experimental results indicate that the performance difference between the two configurations is negligible, with a maximum variation of less than 0.1 seconds, as shown in Figure 2.

For GCC and Clang, there are some differences in the way these two compilers implement auto-vectorization, which may cause performance differences, but experiments show that this difference is negligible as shown in Figure 4.

While our experiments reveal limited benefits from compiler auto-vectorization, we further investigate whether manual RVV code optimization can unlock higher performance for specific query patterns in the next subsection.

Zvl Extension.The *Zvl Extension* is a modular enhancement to the base execution engine that introduces vectorized processing capabilities optimized for heterogeneous hardware platforms. Specifically designed to exploit architectural features such as the RISC-V Vector Extension (RVV), Zvl dynamically adapts its execution strategy at runtime by selecting an appropriate vector length and instruction configuration based on the capabilities of the underlying hardware.

The Spacemit M1 processor recommends a default vector length of 256 bits. To assess the sensitivity of performance to vector length, we performed experiments using compilation configurations targeting 128, 512, 1024, 2048, and 4096 bit vectors. A binary-level

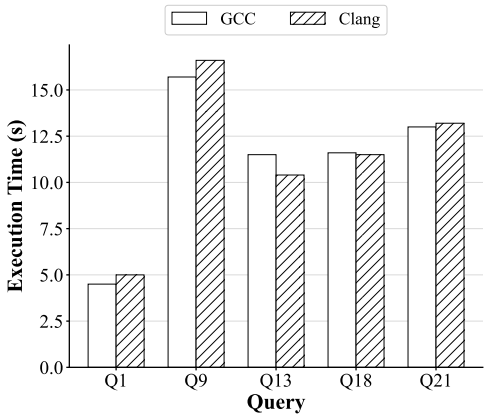


Figure 4: Query execution times of a RISC-V-based database compiled with GCC and Clang On TPC-H SF10.

comparison revealed minimal differences across these configurations, with only 0.15% of the binary content varying. In practical runtime tests, the `zvl` parameter had negligible impact on overall data processing performance, as illustrated in Table 3. This suggests that the compiler effectively adapts to the available vector length, generating near-optimal code regardless of the configured `zvl` value.

Zvl length	Time
zvl128b	5.65s
zvl256b	5.61s
zvl512b	5.61s
zvl1024b	5.61s
zvl2048b	5.61s
zvl4096b	5.61s

Table 3: Average query time across different Zvl vector configurations on TPC-H SF10

Zfh Extension.The *Zfh Extension* enhances the execution engine by enabling efficient computation using half-precision (16-bit) floating-point arithmetic, in line with the RISC-V Zfh specification. This extension targets workloads that are tolerant to reduced precision, such as approximate query processing, machine learning inference, and aggregation over noisy data, where computational throughput and memory bandwidth are critical.

By introducing native support for half-precision operations, Zfh significantly reduces the memory footprint and increases SIMD packing density. The query planner is augmented to detect eligible operations and automatically downcast operands and intermediate results to 16-bit floats when precision constraints allow. At runtime, Zfh leverages hardware-supported conversions and arithmetic instructions to avoid costly emulation overheads. However, we observed that 27% performance degradation(Figure 5b) upon enabling the `Zfh` extension may stem from several architectural and compiler-related factors. First, the activation of `Zfh` may alter the compiler's optimization decisions, potentially resulting in unintended insertion of half-precision floating-point operations and type conversion instructions (e.g., `fcvt.h.s` or `fcvt.s.h`), even when the data is predominantly in single or double precision. Second, the introduction of `Zfh` may increase register pressure, especially in the floating-point register file, thus affecting register allocation and spilling behavior during function calls. This could lead to a higher frequency of memory accesses and degradation of execution efficiency. Third, on in-order RISC-V cores or microarchitectures with limited instruction dispatch capabilities, the inclusion of additional `Zfh` instructions may exacerbate pipeline hazards or structural conflicts, thus reducing overall instruction throughput. Together, these factors may explain the counterintuitive slowdown observed in the database workload when `Zfh` is enabled.

C Extension To evaluate the effect of the RISC-V compressed instruction set (`C` extension), we compiled DuckDB, SQLite and PostgreSQL on RISC-V platforms with and without the extension enabled. As summarized in Figure 3b, enabling the `C` extension reduces the binary size by 24% for DuckDB, 17.8% for PostgreSQL, and 5% for SQLite. These reductions reflect the structural differences between systems, with larger or more modular engines benefiting more. Despite expectations that reduced binary size could alleviate instruction cache pressure, particularly in vectorized engines with dense execution plans, our benchmarks show no measurable runtime performance improvement across tested workloads. This suggests that, under current compiler and hardware configurations, instruction fetch and decode are not dominant bottlenecks, and the primary benefit of the `C` extension remains in reducing the memory footprint rather than execution latency.

D extension.The `D` extension provides hardware acceleration for double-precision floating-point operations. To assess its impact on analytic query performance, we compared the execution times of TPC-H SF10 with and without the `D` extension enabled. As shown in Figure 3c, disabled this extension consistently leads to performance degradation, with an average slowdown of 1 to 2 seconds per query.

Virtual Memory Extension.This part of the test is performed on `QEMU10`. This part involves the two RISC-V extensions `Svpbmt` and `Svnapot`. Enabling these two options does not significantly improve database performance.

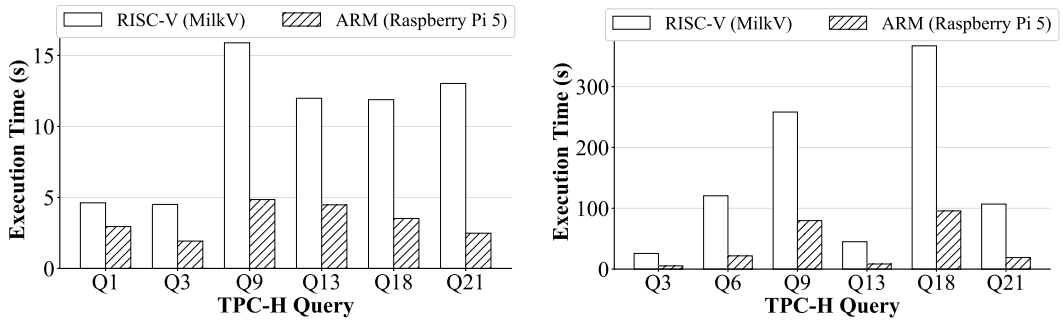
Zknd Extension. The `Zknd` extension introduces specialized RISC-V instructions for accelerating SHA-based cryptographic hash functions. We evaluated its impact on analytical query performance using the TPC-H SF3 benchmark under QEMU-based simulation. As shown in Figure 3d, enabling `Zknd` resulted in consistent performance degradation across all queries. This outcome suggests that SHA acceleration is largely irrelevant for typical database workloads, where hash joins and aggregations rely on non-cryptographic hash functions. The observed slowdown may stem from increased branching overhead, contention on shared execution resources, or suboptimal code generation triggered by the presence of cryptographic instructions. Moreover, simulation artifacts and instruction scheduling changes may exacerbate cache behavior, further limiting overall efficiency.

4.5 Cross-Architecture Performance Comparison

Although the RISC-V architecture has made significant progress in recent years, it still lags far behind the ARM architecture, despite both being based on the RISC design philosophy. As shown in Figure 5, there is a substantial performance gap in the TPC-H benchmark between the Broadcom BCM2712 (Raspberry Pi 5) and the Spacemit M1 (Milk-V Jupiter). The average execution time on the RISC-V platform is approximately 3 to 5 times slower. This disparity is in part due to architectural differences: Spacemit M1 runs at a lower frequency (1.8 GHz) and only supports in-order execution, whereas Broadcom BCM2712 features higher frequencies (2.4Ghz) and support out-of-order execution [1]. Furthermore, the IPC (instructions per cycle) of current RISC-V designs is notably lower than the mainstream CPUs. This performance gap is largely driven by economic factors: increasing IPC typically requires larger chips with more transistors, which in turn reduces power efficiency, increases manufacturing costs, and adds complexity to the design [25]. Given the highly competitive nature of the processor market, most RISC-V vendors have so far prioritized the microcontroller and embedded system segments. They are gradually climbing up the performance ladder, but designing a truly high-performance CPU is a multi-year effort requiring substantial engineering resources. Currently, the RISC-V ISA is undergoing rapid development, with frequent specification updates and extension proposals actively maintained by the community on GitHub [40].

4.6 Case Study: Manual RVV-Aware Query Optimization

Motivated by performance inconsistencies observed in preliminary experiments and to explore the potential of RVV beyond compiler defaults, we implemented a TPC-H scale factor 3 (SF3) benchmark using Apache Arrow to load data from Apache Parquet files with hand-optimized vectorized kernels on the Milk-V Jupiter development board. Our evaluation demonstrates that using the RISC-V Vector Extension (RVV) improves query performance across the benchmark, with speed-ups ranging from 3% to 10×. While Query 4 and Query 6 benefit most significantly, we also observe meaningful acceleration in Query 12 (2.26×) and modest gains in Query 9 (3%), as detailed in Figure 6.



(a) SF10 of DuckDB **(b) SF3 of TiDB**

Figure 5: Query execution time comparison on TPC-H running on Arm-based Broadcom BCM2712 (Raspberry Pi 5) and RISC-V–based Spacemit M1 (Milk-V Jupiter).

A closer examination of representative queries reveals the practical value and limitations of RVV-aware optimization. Query 1 benefits from vectorized arithmetic operations to compute derived columns (e.g., `discount_price`, `taxed_price`) and vectorized aggregation for `SUM`/`AVG` metrics. Query 4 leverages RVV for parallelized date comparisons (e.g., `order_date < '1993-10-01'`), while Query 6 achieves its 10× speedup through vectorized conditional aggregation in both selection and accumulation stages. For Query 12, RVV accelerates the evaluation of complex shipping rule predicates through vectorized string matching and date arithmetic. Query 9 shows more modest gains (3%) due to its dependence on hash joins that benefit less from our current vectorization approach — suggesting join algorithms as future optimization targets.

These results demonstrate that explicit RVV utilization can significantly outperform compiler defaults, though optimization efficacy varies with query characteristics. Consistent improvements highlight the importance of low-level architectural awareness in query execution in RISC-V, particularly for compute-intensive operations where RVV delivers 1.03–10× speedups in our benchmark.

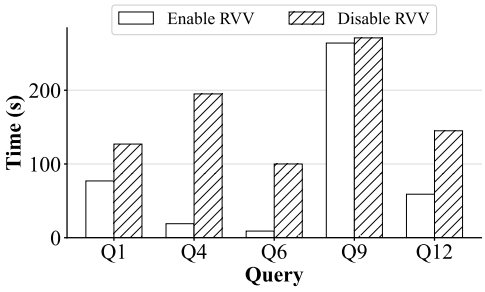


Figure 6: Execution time comparison for TPC-H queries Q1, Q4, Q6, Q9, and Q12 at scale factor 3 (SF3) using Apache Arrow with Apache Parquet input

5 DISCUSSION

Our experimental evaluation reveals the nuanced relationship between RISC-V architectural features and database system performance. Although baseline benchmarks illustrate the feasibility of running relational databases on RISC-V, a deeper analysis of the characteristics at the ISA level uncovers both latent performance opportunities and practical limitations. We now summarize the key insights gained and outline directions for future research.

5.1 Key Findings

Our study yields several important insights into the viability and performance of relational databases on RISC-V:

- **SQLite exhibits the highest out-of-the-box compatibility** among the tested databases, compiling and executing successfully on RISC-V platforms without modifications. PostgreSQL shows the strongest community and upstream support, as reflected in the active maintenance of official RISC-V packages by several major Linux distributions. MariaDB and TiDB provide partial but promising support. DuckDB can be compiled with minor attention to compiler parameters. However, the lack of robust and widely adopted CI solutions targeting RISC-V presents a practical barrier to sustained support, as it discourages developers from actively maintaining cross-architecture compatibility.
- **Most RISC-V ISA extensions have a limited impact on mature database systems**. Enabling the Vector Extension (`V`) provides less than 5% improvement on TPC-H SF10 using default compilation pipelines, with no significant variation between GCC and LLVM. The `D` extension offers modest improvements (1–2 seconds reduction) due to hardware-accelerated double precision arithmetic. The `Zvl` vector-length extensions show negligible effect on performance in current workloads and compilers.
- **Some extensions can negatively affect performance**. Activating the `Zknd` cryptographic extension reduces performance by up to 32%, likely due to code generation overheads and immature support in back-end optimizers. Likewise, the `Zfh` half-precision extension increases the execution time in several queries, as shown in Figure 5b.

- **The compressed instruction set (`C`) significantly reduces binary size**—by up to 24%—without impacting runtime performance. This makes it particularly beneficial for memory-constrained deployments such as embedded systems.
- **Manually optimized RVV implementations provide substantial performance gains**. In our case study (subsection 4.6), handwritten vectorized kernels using RVV achieved up to 10× speedups, especially for filter and aggregation-heavy queries such as Query 4 and Query 6. This highlights the performance ceiling imposed by current compiler auto-vectorization and the benefit of hardware-aware tuning.

These findings suggest that, while RISC-V is a promising target for general-purpose database systems, realizing its full potential requires fine-grained architectural awareness and low-level optimization beyond what current compilers automatically provide.

5.2 Future Directions

Building on our evaluation of general-purpose RISC-V platforms for relational database workloads, we identify several promising directions for future research.

(1) Supporting Embedded and Edge-Oriented Extensions. To extend the support of the database to constrained environments, future work should explore RISC-V extensions such as `Zfinx`, `Zdinx`, and `Zhinx`, which enable floating-point operations using integer registers. These extensions offer reduced hardware complexity and binary size, aligning well with edge computing and IoT deployments. Compiler support in GCC and LLVM is already progressing, and empirical evaluation on embedded-class RISC-V chips would help quantify their practical benefits.

(2) Co-Designing Custom ISA Extensions for Databases. The modularity of RISC-V opens opportunities to tailor the ISA for database acceleration. Potential directions include:

- **Query planning and execution optimizations:** Introduce custom instructions for filtering, aggregation, and join primitives.
- **Memory hierarchy tuning:** Explore cache-aware or memory-efficient layouts in conjunction with hardware hints.
- **Vector-aware JIT support:** Build JIT engines that emit RVV-adaptive code paths depending on runtime `VLEN` and microarchitecture capabilities.

Tools such as Spike [34] and Sail [2] enable prototyping such changes, and hardware simulators can assist in cost-benefit analysis before silicon implementation.

Together, these directions aim to deepen the synergy between RISC-V's hardware flexibility and the evolving needs of relational data processing, especially as the architecture enters new deployment domains.

6 RELATED WORK

RISC-V Architecture. In 2023, Perfxlab investigated the adaptability of the database to RISC-V on the SG2042 processor, but only tested compatibility and did not run performance-related tests [28]. Gao et al. analyzed the possibility of running big data applications on the RISC-V architecture [10]. Jihwan Lim et al. tried to extend ISA to increase the speed of memory data processing in embedded devices, achieving a 34.4% speed increase and 18% energy reduction [16]. Heng Lin et al. tried to use the Rust language to implement the early version of the RISC-V V Extension library, speeding up the basic linear algebra subroutines(BLAS) by 1.3 to 2.31 times [17]. Quentin et al. studied the problems caused by the X86 and RISC-V architectures in JIT compiler porting [8]. Konstantin et al. tried to analyze the efficiency of RISC-V ISA on ANN algorithms [35].

RISC Type Architecture. As RISC-based architectures, both ARM and RISC-V share similar design philosophies, emphasizing simplicity and efficiency. They are frequently compared in terms of performance, power consumption, and hardware support. Gruber et al. evaluated the Umbra database system on an AArch64-based platform on TPC-H Benchmark, conducting a detailed comparison with x86-64 [12]. Their analysis highlights key architectural differences, particularly in memory ordering semantics, memory alignment requirements, arithmetic operations, instruction selection, and the handling of immediate operands.

Hardware Accelerated RDBMS. Recent studies have explored hardware acceleration to improve RDBMS performance. Mancini et al. investigated GPU acceleration to optimize join operations on large-scale multi-table datasets [20] , while another work by Riccardo Mancini's team employed FPGA to improve PostgreSQL's read performance [19]. Furthermore, Perach et al. demonstrated the potential of processing-in-memory (PIM) architectures to accelerate read operations in relational databases [27]. These approaches collectively highlight the growing trend of leveraging specialized hardware to address performance bottlenecks in RDBMS.

7 CONCLUSION

In this paper, we present an in-depth evaluation of relational database systems on RISC-V platforms, revealing how architectural features and toolchain configurations influence performance. By assessing database engines across a range of RISC-V ISA extensions and compiler toolchains, we show that, while RISC-V is increasingly capable of supporting modern data processing workloads, its efficiency depends strongly on hardware-aware tuning and compiler maturity. Our results indicate that many standard ISA extensions contribute little to performance in default settings, while features like compressed instructions benefit memory-constrained deployments. Manual optimization with RVV yields substantial gains, highlighting the current limitations of auto-vectorization. These findings suggest a need for closer integration between database design and RISC-V architecture evolution, particularly in embedded and edge computing scenarios, where custom ISA co-design and improved toolchain support can unlock greater efficiency.

REFERENCES

[1] Arm Ltd. 2025. Cortex-A76 | Laptop-class Performance with Mobile Efficiency. https://www.arm.com/products/silicon-ip-cpu/cortex-a/cortex-a76. Accessed: 2025-07-03.

[2] Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. 2019. ISA semantics for ARMv8-a, RISC-v, and CHERI-MIPS. *Proc. ACM Program. Lang.* 3, POPL, Article 71 (Jan. 2019), 31 pages. https://doi.org/10.1145/3290384

[3] Krste Asanović, David Patterson, et al. 2016. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1. https://people.eecs.berkeley.edu/~krste/papers/riscv-spec-v2.1.pdf. RISC-V Foundation.

[4] Nick Clifton. 2018. GCC 8.2 Released with RISC-V Support. https://lists.gnu.org/archive/html/info-gnu/2018-09/msg00001.html. GNU Project Announcement.

[5] Debian Developers. 2025. RISC-V is now a release architecture for Debian 13 (trixie). https://lists.debian.org/debian-devel-announce/2025/05/msg00004.html. Accessed: 2025-05-21.

[6] Debian Developers. 2025. RISC-V is now an official Debian architecture. https://lists.debian.org/debian-devel-announce/2025/05/msg00004.html. Accessed: 2025-05-21.

[7] MariaDB Developers. 2025. Fix building with Clang and GCC on RISC-V. https://github.com/MariaDB/server/commit/05be1865a9. Commit 05be186, accessed: 2025-06-01.

[8] Quentin Ducasse, Guillermo Polito, Pablo Tesone, Pascal Cotret, and Loïc Lagadec. 2022. Porting a JIT Compiler to RISC-V: Challenges and Opportunities. In *Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes* (Brussels, Belgium) *(MPLR '22)*. Association for Computing Machinery, New York, NY, USA, 112–118. https://doi.org/10.1145/3546918.3546924

[9] DuckDB Contributors. 2024. Feature Request: Add RISC-V support Docker Test for DuckDB. https://github.com/duckdb/duckdb/discussions/16462. https://github.com/duckdb/duckdb/discussions/16462 GitHub Discussion #16462.

[10] Yue Gao, Enfang Cui, and Tianzheng Li. 2025. A Review of Big Data Applications Based on RISC-V. In *Proceedings of the 2024 3rd International Conference on Algorithms, Data Mining, and Information Technology (ADMIT '24)*. Association for Computing Machinery, New York, NY, USA, 316–320. https://doi.org/10.1145/3701100.3701165

[11] Go Authors. 2022. RISC-V: proposal to move riscv64 port to first class port. https://github.com/golang/go/issues/52644. Accessed: 2025-05-21.

[12] Ferdinand Gruber, Maximilian Bandle, Alexis Engelke, Thomas Neumann, and Jana Giceva. 2023. Bringing Compiling Databases to RISC Architectures. *Proc. VLDB Endow.* 16, 6 (Feb. 2023), 1222–1234. https://doi.org/10.14778/3583140.3583142

[13] Richard Hipp. 2021. Forum post: Optimizing indexes with WHERE clause expressions. https://sqlite.org/forum/forumpost/92a85830dd5e5277d054fafe56e10c9bdc198a39a114c9d80aa65e7e15d20d44. Accessed: 2025-07-03.

[14] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, Wan Wei, Cong Liu, Jian Zhang, Jianjun Li, Xuelian Wu, Lingyu Song, Ruoxi Sun, Shuaipeng Yu, Lei Zhao, Nicholas Cameron, Liquan Pei, and Xin Tang. 2020. TiDB: a Raft-based HTAP database. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3072–3084. https://doi.org/10.14778/3415478.3415535

[15] Wenqi Jiang, Dario Korolija, and Gustavo Alonso. 2023. Data Processing with FPGAs on Modern Architectures. In *Companion of the 2023 International Conference on Management of Data* (Seattle, WA, USA) *(SIGMOD '23)*. Association for Computing Machinery, New York, NY, USA, 77–82. https://doi.org/10.1145/3555041.3589410

[16] Jihwan Lim, Jeonghun Son, and Hoyoung Yoo. 2024. Efficient Processing-in-Memory System Based on RISC-V Instruction Set Architecture. *Electronics* 13, 15 (2024). https://doi.org/10.3390/electronics13152971

[17] Heng Lin, Piyo Chen, Yuan-Shin Hwang, and Jenq-Kuen Lee. 2019. Devise Rust Compiler Optimizations on RISC-V Architectures with SIMD Instructions. In *Workshop Proceedings of the 48th International Conference on Parallel Processing* (Kyoto, Japan) *(ICPP Workshops '19)*. Association for Computing Machinery, New York, NY, USA, Article 13, 7 pages. https://doi.org/10.1145/3339186.3339193

[18] LLVM Project. 2025. RISC-V Usage in LLVM. https://llvm.org/docs/RISCVUsage.html. Accessed: 2025-05-21.

[19] Divya Mahajan, Joon Kyung Kim, Jacob Sacks, Adel Ardalan, Arun Kumar, and Hadi Esmaeilzadeh. 2018. In-RDBMS hardware acceleration of advanced analytics. *Proc. VLDB Endow.* 11, 11 (July 2018), 1317–1331. https://doi.org/10.14778/3236187.3236188

[20] Riccardo Mancini, Srinivas Karthik, Bikash Chandra, Vasilis Mageirakos, and Anastasia Ailamaki. 2022. Efficient Massively Parallel Join Optimization for Large Queries. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 122–135. https://doi.org/10.1145/3514221.3517871

[21] Prashanth Menon, Todd C. Mowry, and Andrew Pavlo. 2017. Relaxed operator fusion for in-memory databases: making compilation, vectorization, and prefetching work together at last. *Proc. VLDB Endow.* 11, 1 (Sept. 2017), 1–13. https://doi.org/10.14778/3151113.3151114

[22] Milk-V. 2025. Milk-V Jupiter - Detail. https://milkv.io/jupiter. Available at https://milkv.io/jupiter.

[23] MySQL Bugs Database. 2020. MySQL Bug #100356: Compressed columns are not used for index statistics. https://bugs.mysql.com/bug.php?id=100356 Accessed: 2025-06-01.

[24] Haoran Ning, Bocheng Han, Zhengyi Yang, Kongzhang Hao, Miao Ma, Chunling Wang, Boge Liu, Xiaoshuang Chen, Yu Hao, Yi Jin, Wanchuan Zhang, and Chengwei Zhang. 2024. Exploring Simple Architecture of Just-in-Time Compilation in Databases. In *Web and Big Data*, Wenjie Zhang, Anthony Tung, Zhonglong Zheng, Zhengyi Yang, Xiaoyang Wang, and Hongjie Guo (Eds.). Springer Nature Singapore, Singapore, 504–514.

[25] Omdia. 2023. *RISC-V Processors Report.* Technical Report. Omdia. https://omdia.tech.informa.com/-/media/tech/omdia/brochures/ai/risc-v-processors-report.aspx Accessed: 2025-05-18.

[26] Che Pan. 2023. Chip War: Chinese Scientists Vow to Launch Breakthrough RISC-V Open-source CPU by 2025. *South China Morning Post* (dec 2023). https://www.scmp.com/tech/tech-war/article/3293610/chip-war-chinese-scientists-vow-launch-breakthrough-risc-v-open-source-cpu-2025 Accessed: 2025-05-19.

[27] Ben Perach, Ronny Ronen, and Shahar Kvatinsky. 2023. Accelerating Relational Database Analytical Processing with Bulk-Bitwise Processing-in-Memory. In *2023 21st IEEE Interregional NEWCAS Conference (NEWCAS)*. 1–5. https://doi.org/10.1109/NEWCAS57931.2023.10198122

[28] PerfXLab. 2023. RISC-V Public Beta Platform Release. RISC-V International Blog. https://riscv.org/blog/2023/08/risc-v-public-beta-platform-release Accessed: 2025-05-19.

[29] PostgreSQL Global Development Group. 2025. PostgreSQL 16 Documentation: 17.6. Supported Platforms. https://www.postgresql.org/docs/16/supported-platforms.html. Accessed: 2025-05-31.

[30] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1981–1984. https://doi.org/10.1145/3299869.3320212

[31] Red Hat. 2024. Red Hat Partners with SiFive for RISC-V Developer Preview for Red Hat Enterprise Linux 10. https://www.redhat.com/en/blog/red-hat-partners-with-sifive-for-risc-v-developer-preview-for-red-hat-enterprise-linux-10. Accessed: 2025-05-27.

[32] RISC-V International. 2021. The RISC-V Vector Extension, Version 1.0. https://github.com/riscv/riscv-v-spec. Accessed: 2025-05-19.

[33] RISC-V Software Development Mail List. 2023. Re: Status of Timestamp Counter on RISC-V. https://groups.google.com/a/groups.riscv.org/g/sw-dev/c/2-u-c3kyZlc/m/awYsLfQ_BwAJ. Accessed: 2025-05-21.

[34] RISC-V Software Source. 2025. riscv-isa-sim: Spike RISC-V ISA Simulator. https://github.com/riscv-software-src/riscv-isa-sim. Accessed: 2025-05-09.

[35] Konstantin Rumyantsev, Pavel Yakovlev, Andrey Gorshkov, and Andrey P. Sokolov. 2024. RISC-V RVV efficiency for ANN algorithms. arXiv:2407.13326 [cs.LG] https://arxiv.org/abs/2407.13326

[36] Rust Embedded WG. 2025. riscv – Rust crate for RISC-V specific functionality. https://docs.rs/riscv. Accessed: 2025-05-21.

[37] PingCAP TiDB Team. 2023. My machine is riscv64, how do I deploy and install tidb? https://github.com/pingcap/tidb/issues/46088. GitHub Issue #46088.

[38] Transaction Processing Performance Council (TPC). 2023. TPC-H Benchmark. http://www.tpc.org/tpch/. http://www.tpc.org/tpch/ Version 3.0.1.

[39] Andrew Waterman and Krste Asanovic. 2017. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2. https://riscv.org/specifications/.

[40] Andrew Waterman, Krste Asanović, and RISC-V International. 2023. The RISC-V Instruction Set Manual. GitHub repository. https://github.com/riscv/riscv-isa-manual Commit: d6a7bb3, 2023-12-13.

[41] XiangShan Project. 2025. XiangShan Biweekly Report #69. https://docs.xiangshan.cc/zh-cn/latest/blog/2025/02/03/biweekly-69-en/. Accessed: 2025-05-19.