Literature Survey: Vector Instruction Sets

1st Shao-Chien Chiang
Dept. of Computer and Science Engr.
Texas A&M University
College Station, TX

2nd Hsin-An Ko
Dept. of Electrical and Computer Engr.
Texas A&M University
College Station, TX

3rd Emily Chang
Dept. of Electrical and Computer Engr.
Texas A&M University
College Station, TX

4th Pin-Tzu Huang
Dept. of Electrical and Computer Engr.
Texas A&M University
College Station, TX

***Abstract*—Vector instruction sets have played a crucial role in high-performance computing, enabling efficient parallel processing across various architectures. This paper explores the evolution and design of vector instruction sets, focusing on three major architectures: x86 AVX, ARM SVE, and RISC-V Vector Extensions (RVV). The study examines their architectural differences, performance implications, and trade-offs in terms of flexibility, power efficiency, and memory access optimizations. While AVX provides fixed vector widths with explicit mask registers for selective operations, SVE introduces scalable vector lengths and predicate-based masking for enhanced adaptability. RVV further extends flexibility by allowing variable-length vector execution based on hardware constraints, making it suitable for domain-specific applications. The paper also evaluates the performance impact of vector extensions in fields such as high-performance computing, artificial intelligence, and multimedia processing. Future challenges, including memory bottlenecks, compiler optimizations, and integration with emerging hardware accelerators, are discussed. The findings highlight the continuous evolution of vector instruction sets and their significance in modern computing architectures.**

I. Introduction

After the introduction of the iconic CRAY-1 [2] machine in 1976, vector processors dominated supercomputers, but declined in the 1990s. However, their design principles continued to influence future ISA extensions. As the demands for multimedia and graphics grew, many commodity microprocessors introduced SIMD extensions to support these computations [3]. These extensions consist of short vector instructions that operate on packed data in registers. Early implementations used existing microarchitecture with insignificant changes [6]. They partitioned the ALU to support subword operations, or remapped floating-point registers to new SIMD registers. Those extensions also included MMX [4] for Intel x86, VIS [5] for UltraSPARC, and MDMX for MIPS. While overloading registers maintained compatibility with existing OS context switch routines, it also hindered simultaneous use of floating-point and SIMD operations. It increased the complexity of both hardware and software to accommodate shared registers for future extensions. SSE [13] for x86 and AltiVec [14] for PowerPC introduced independent 128-bit registers to address this issue. SIMD extensions came with several issues [9], [10]. Some of these issues still exist these days. The first challenge is data alignment. Hardware needed mechanisms to handle SIMD instructions that accessed data across page boundaries. For software, we had to handle situations where the data length was not a multiple of the vector length, or where unaligned memory accesses could cause significant performance impact. Another issue was that it was difficult to find a general way to optimize codes because optimizations were dependent on the features of each extension. Compilers could not generate efficient vectorization instructions. To fully utilize the power of SIMD, many programmers wrote their vector codes in handwritten assembly. Research [11] showed that only a small portion of programs executed SIMD calculations, while 75–80SVE [1] for ARM introduced several innovations to address these problems. By leaving the vector length as an implementation choice, its program model allowed software to run without needing to recompile to support the available vector length. SVE also introduced the ability to perform predicted operations. This prediction was useful in reducing the overhead and supporting instructions mentioned earlier. Similarly, AVX-512 from x86 also supports predicated instructions. The introduction of first-fault mechanisms enabled fault-tolerant speculative vectorization, suppressing memory faults and indicating which elements were unsuccessfully loaded. The new horizontal operations solved the problems of dependencies across vectors. These designs also improved compiler auto-vectorization. Meanwhile, the RISC-V vector extension introduced a distinct design. Its vector length could dynamically vary at runtime. In addition to their initial use in multimedia and graphics, vector extensions can enhance applications across various fields, with designs adopting concepts from traditional vector processors, such as variable vector lengths and predicated instructions.

II. Architecture Design and Implementation

In today's mainstream computer architectures, vector instruction sets can be broadly categorized into three major types: the AVX instruction set, which was introduced early and continues to be updated for the x86 architecture; the more recently introduced SVE instruction set for the ARM architec-

ture; and the vector instruction set provided by RISC-V, the most popular open architecture, enabling broader application scenarios.

The following discussion will be divided into three parts:

Part A. Compare the design of vector instruction sets across different architectures. Part B. Examine architectural design features. Part C. Analyze the trade-offs in instruction set design.

A. Compare the design of vector instruction sets across different architectures.

1) x86 AVX vs. ARM SVE: One of the fundamental differences between AVX and SVE lies in their approach to vector width. AVX employs fixed vector widths of 128-bit (AVX), 256-bit (AVX2), and 512-bit (AVX-512) [13]. In contrast, SVE supports scalable vector lengths ranging from 128-bit to 2048-bit, allowing runtime adaptation and greater flexibility [15]. This scalable approach in SVE enables more efficient use of hardware resources, as the same binary can run on different hardware implementations without modification.

Masking mechanisms further differentiate AVX and SVE. AVX-512 incorporates explicit mask registers (K0-K7) to enable selective operations on vector elements, which enhances performance in specific computational workloads [13]. However, SVE takes a different approach by utilizing predicate registers for lane masking, which offers more dynamic and flexible control over which elements participate in computation [16] This design simplifies compiler optimizations and increases execution efficiency.

Memory access is another crucial aspect where the two architectures diverge. AVX supports explicit gather and scatter operations, but it requires multiple instructions for optimal utilization, making it somewhat complex in certain workloads [12]. In contrast, SVE integrates gather-load and scatter-store operations with predicate-based control, significantly improving memory efficiency [16]. This built-in flexibility enables better handling of non-contiguous memory accesses, which is particularly beneficial for high-performance computing applications.

2) RISC-V Vector Extensions (RVV): RISC-V Vector Extensions (RVV) introduce a vector-length-agnostic (VLA) approach similar to SVE, where vector width is hardware-dependent and can scale dynamically [17]. This feature allows different implementations to optimize vector execution based on available resources, enhancing performance across a variety of hardware configurations.

Unlike AVX and SVE, which define specific vector widths, RVV provides a high level of customizability. Implementations can define specific vector register widths and instruction subsets, making it highly adaptable for domain-specific applications such as embedded systems and AI accelerators [17]. This flexibility ensures that RISC-V processors can cater to diverse application needs while maintaining efficiency.

Memory access and masking in RVV share similarities with SVE. It supports various access patterns, including strided, indexed, and unit-stride memory operations, with built-in masking mechanisms to control element participation [17]. This capability enhances the efficiency of vectorized operations and simplifies programming, particularly for workloads that require irregular memory access patterns.

B. Examine architectural design features.

1) Impact of Vector Width on Performance: Increasing vector width significantly boosts computational throughput but also introduces challenges in power consumption and architectural complexity. Studies show that expanding vector width in SVE from 128-bit to 512-bit improves performance by up to 38% while reducing energy consumption by 23% [15]. However, beyond 512-bit, the benefits diminish due to memory bandwidth limitations, which hinder further performance gains [16].

2) Hardware Implementation of Variable-Length Vectors: The implementation of vector execution also differs across these architectures. AVX relies on fixed-length execution units, requiring separate ALUs for 128-bit, 256-bit, and 512-bit operations. This design increases chip area and power consumption, particularly when handling mixed-width operations [14]. In contrast, SVE and RVV utilize a single vector unit that dynamically adapts to available vector width, reducing hardware redundancy and improving efficiency [16]

C. Analyze the trade-offs in instruction set design.

1) Power Consumption: Power efficiency is a critical consideration in vector instruction design. AVX consumes substantial power due to its wide registers and fixed-length execution units. AVX-512, in particular, is known to reduce CPU clock speeds under heavy load to mitigate power draw, which can impact performance in sustained workloads [14]. On the other hand, SVE and RVV offer dynamic vector length adaptation, enabling better power efficiency by optimizing resource usage based on workload demands [15].

2) Latency and Throughput Considerations: From a latency and throughput perspective, AVX provides high throughput but incurs overhead in mixed-width operations, requiring additional processing steps [12]. SVE, with its flexible masking and integrated gather/scatter support, reduces latency and improves computational efficiency [16]. Meanwhile, RVV balances flexibility with instruction complexity, making its performance heavily dependent on specific hardware implementations [17].

III. Performance and Application Analysis of Vector Instruction Sets

A. Vector Instruction Sets Across Different Domains

RISC-V Vector Extension (RVV) and Arm Scalable Vector Extension (SVE), provide only one dimensional stride and random memory accesses. While this is sufficient for typical vector engines, it fails to effectively utilize the large Single Instruction, Multiple Data (SIMD) widths of in cache vector engines. This is because mobile data-parallel kernels expose limited parallelism across a single dimension. Therefore, vector instruction sets play a crucial role in modern

computing architectures. By enabling parallel data processing, it improves computational efficiency, and accelerating various applications. These instruction sets have been widely adopted in high-performance computing (HPC), machine learning, and other computationally intensive domains [18].

For instance, multi-dimensional vector ISA Extension (MVE) for mobile in-cache computing uses dimension-level masked execution to reduce overhead in handling irregular parallelism. Additionally, it abstracts cache geometry and data layout, simplifying programming and enhancing efficiency to achieves high SIMD utilization (60% vs. 23% in RVV) and reduces instruction overhead.

B. Performance Optimization in Different Application Scenarios

High-Performance Computing (HPC) provides libraries such as Basic Linear Algebra Subprograms (BLAS) which benefit from optimized vector processing. For instence, LINPACK-PC, a C implementation of the LINPACK benchmark, is available from Netlib9. This benchmark application makes use of several BLAS level 1 and 2 operations. It operates on matrices and vectors of dimensions up to 200x201. LINPACK-PC performs a certain sequence of operations in a loop for 5 seconds, and calculates the achieved MFLOPS from the number of loop iterations [19].

C. Performance Evaluation Methods

Gem5-AVX, an extended version of the Gem5 simulator that enables simulating recent x86 SIMD extensions, especially targeted for high performance computing (HPC). Gem5 is an open-source, event-driven computer architecture simulator used for modeling and simulating modern computer systems. Such as Intel AVX enhances the performance of SIMD instruction sets in prior generations by using a new instruction encoding scheme called the vector extension prefix (VEX). And it provides improved features that surpass those offered by previous 128-bit SIMD extentions [20].

IV. Future Developments and Technical Challenges

In this section, we focus on the trends and technical challenges of vector extensions in high-performance computing (HPC), machine learning, and embedded systems, followed by a quick summary of potential research directions in the future.

A. Review The Past Trends in Vector Instruction Sets and Processors

1) Trends in Vector Instruction Sets and ArchitecturesRoles of CPUs, GPUs, and Coprocessors [17] [21]: The evolution from Fujitsu VPP to ARM SVE and RISC-V highlights the importance of SIMD technology in parallel computing. Early SIMD extensions focused on graphics and multimedia tasks. However, as AI and other emerging technologies advanced, traditional CPUs encountered performance bottlenecks. This challenge spurred the adoption of heterogeneous architectures where CPUs manage low-latency control tasks, while GPUs and coprocessors handle data-parallel operations. This division of labor optimizes overall system performance by leveraging each processor's strengths for specific workloads.

2) ARM SVE Development [22]: ARM Scalable Vector Extension (SVE) boosts HPC and machine learning performance through variable vector lengths, providing flexibility across applications and platforms. SVE reduces software porting costs while maintaining performance. The latest version, SVE2, further optimizes FFT, matrix operations, and multimedia encoding.

3) RISC-V Vector Extensions [17]: RISC-V RVV offers modular and customizable SIMD instructions, supporting 8-, 16-, and 32-bit operations, making it suitable for embedded, IoT, and server applications. However, RISC-V Hypervisor extensions need improvements in I/O and virtualization. SiFive P800 processors, supporting RVV 1.0 and Vector Crypto, enhance performance in vector and cryptographic operations.

B. Challenges from Emerging Applications

1) Precision and Efficiency Requirements in AI Computing [23]: Artificial intelligence (AI) computing, especially deep learning, faces a trade-off between precision and efficiency. For example, in structured sparse matrix multiplication (SpMM), critical parameters such as data distribution, locality, and loop unrolling directly affect performance. To address these challenges, [23] proposed several optimization strategies, including adjusting floating-point storage formats, designing efficient vectorized algorithms, and leveraging custom hardware accelerators.

2) Memory Bottlenecks and Energy Efficiency [24]: Performance discrepancies across different CPU architectures in vectorized hash table applications are closely related to memory access patterns and data locality. As vector operations process large datasets simultaneously, memory bandwidth often becomes a performance bottleneck. Vector instruction sets such as ARM SVE play a crucial role in mitigating memory bottlenecks and improving energy efficiency. Strategies such as optimizing data layout in memory, employing caching and prefetching techniques, and reducing memory access frequency can effectively enhance vector computation performance. However, different vectorized hashing schemes (e.g., VLP, VFP, and BBC) exhibit varying scalability in multi-threaded environments, necessitating careful evaluation.

C. Collaboration of Vector Instruction Sets with Other Acceleration Technologies [18] [25]

To further improve overall performance, CPUs, GPUs, and dedicated coprocessors must collaborate. GPUs accelerate graphics processing and deep learning tasks, while dedicated coprocessors optimize specific applications. Multi-dimensional vector ISA extensions (MVX) can significantly enhance in-cache computing efficiency on mobile devices [18]. VersaTile, an innovative heterogeneous multi-core architecture [25], assigns traditional CPU cores to low-latency control tasks and application (AP) cores to high-throughput, data-parallel

tasks. These cores are interconnected via a tile-based mesh, facilitating flexible resource scheduling and maximizing the advantages of different cores to improve overall performance and energy efficiency.

V. Future Directions

A. Customized Vector Extensions for Specific Applications

Open architectures like RISC-V facilitate tailored vector instructions, such as accelerating cryptographic operations through specialized extensions, enhancing performance for targeted workloads.

B. Integration with Emerging Memory Technologies

Advanced memory technologies, such as High Bandwidth Memory (HBM) and 3D-stacked memory, mitigate vector computation bottlenecks by offering higher bandwidth. Future research should focus on optimizing compatibility between vector instruction sets and these memory solutions.

C. Advancement of Automatic Vectorization Compilers

Enhanced automatic vectorization compilers can reduce software development costs by converting traditional code to vectorized code. Future improvements should broaden support for diverse vector instruction sets and application scenarios. In conclusion, the future development of vector instruction sets faces significant technical challenges, including the need for highly flexible designs suitable for diverse applications and seamless integration with other acceleration technologies. Vector extensions are indispensable in HPC, AI, and embedded systems. Integrating advancements in RISC-V, ARM SVE, and other related technologies will play a crucial role in advancing high-performance computing.

References

[1] N. Stephens et al., "The ARM Scalable Vector Extension," in IEEE Micro, vol. 37, no. 2, pp. 26-39, Mar.-Apr. 2017, doi: 10.1109/MM.2017.35.

[2] J. Kolodzey, "CRAY-1 Computer Technology," in IEEE Transactions on Components, Hybrids, and Manufacturing Technology, vol. 4, no. 2, pp. 181-186, June 1981, doi: 10.1109/TCHMT.1981.1135787.

[3] Roger Espasa, Mateo Valero, and James E. Smith. 1998. Vector architectures: past, present and future. In Proceedings of the 12th international conference on Supercomputing (ICS '98). Association for Computing Machinery, New York, NY, USA, 425–432. https://doi.org/10.1145/277830.277935

[4] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," in IEEE Micro, vol. 16, no. 4, pp. 42-50, Aug. 1996, doi: 10.1109/40.526924.

[5] L. Kohn, G. Maturana, M. Tremblay, A. Prabhu and G. Zyner, "The visual instruction set (VIS) in UltraSPARC," Digest of Papers. COMPCON'95. Technologies for the Information Superhighway, San Francisco, CA, USA, 1995, pp. 462-469, doi: 10.1109/CMPCON.1995.512423.

[6] R. B. Lee, "Multimedia extensions for general-purpose processors," 1997 IEEE Workshop on Signal Processing Systems. SiPS 97 Design and Implementation formerly VLSI Signal Processing, Leicester, UK, 1997, pp. 9-23, doi: 10.1109/SIPS.1997.625683.

[7] S. Thakkur and T. Huff, "Internet Streaming SIMD Extensions," in Computer, vol. 32, no. 12, pp. 26-34, Dec. 1999, doi: 10.1109/2.809248.

[8] K. Diefendorff, P. K. Dubey, R. Hochsprung and H. Scale, "AltiVec extension to PowerPC accelerates media processing," in IEEE Micro, vol. 20, no. 2, pp. 85-95, March-April 2000, doi: 10.1109/40.848475.

[9] T. M. Conte et al., "Challenges to combining general-purpose and multimedia processors," in Computer, vol. 30, no. 12, pp. 33-37, Dec. 1997, doi: 10.1109/2.642799.

[10] M. Hassaballah, S. Omran and Y. B. Mahdy, "A Review of SIMD Multimedia Extensions and their Usage in Scientific and Engineering Applications," in The Computer Journal, vol. 51, no. 6, pp. 630-649, Nov. 2008, doi: 10.1093/comjnl/bxm099.

[11] D. Talla, L. K. John and D. Burger, "Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements," in IEEE Transactions on Computers, vol. 52, no. 8, pp. 1015-1031, Aug. 2003, doi: 10.1109/TC.2003.1223637.

[12] H. Amiri and A. Shahbahrami, "SIMD programming using Intel vector extensions," J. Parallel Distrib. Comput., vol. 135, pp. 83–100, Jan. 2020. doi: 10.1016/j.jpdc.2019.09.012.

[13] D. Kusswurm, "Modern X86 Assembly Language Programming: Covers X86 64-bit, AVX, AVX2, and AVX-512", Berkeley, CA: Apress, 2018. doi: 10.1007/978-1-4842-3005-6.

[14] P. Gepner, V. Gamayunov, and D. L. Fraser, "Early performance evaluation of AVX for HPC", International Conference on Computational Science, ICCS 2011.

[15] T. Odajima, Y. Kodama, and M. Sato, "Power Performance Analysis of ARM Scalable Vector Extension",

[16] Yu. Kodama, T. Odajima, M. Matsuda, M. Tsuji, J. Lee, and M. Sato, "Preliminary Performance Evaluation of Application Kernels using ARM SVE with Multiple Vector Lengths", in 2017 IEEE International Conference on Cluster Computing (CLUSTER), 2017, pp. [page numbers]. doi: 10.1109/CLUSTER.2017.93.

[17] E. CUI ,T. LI, and Q. WE, "RISC-V Instruction Set Architecture Extensions: A Survey", IEEE Access, vol. PP, no. 99, pp. 1-1, Jan. 2023. doi: 10.1109/ACCESS.2023.3246491.

[18] A. Khadem, D. Fujiki, H. Chen, Y. Gu, N. Talati, S. Mahlke, and R. Das, "Multi-dimensional vector ISA extension for mobile in-cache computing," arXiv preprint arXiv:2501.09902, 2025. [Online]. Available: https://arxiv.org/abs/2501.09902

[19] C. Fibich et al., "Evaluation of Open-Source Linear Algebra Libraries in Embedded Applications," 2019 8th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2019, pp. 1-6, doi: 10.1109/MECO.2019.8760041.

[20] S. Lee, Y. Kim, D. Nam and J. Kim, "Gem5-AVX: Extension of the Gem5 Simulator to Support AVX Instruction Sets," in IEEE Access, vol. 12, pp. 20767-20778, 2024, doi: 10.1109/ACCESS.2024.3359296.

[21] D. Vută, I. C. Antofi, C. B. Ciobanu, and C. Z. Kertész, "SIMD Extensions - A Historical Perspective," in Proc. IEEE Int. Symp. Design Technol. Electron. Packag. (SIITME), Sibiu, Romania, Oct. 2024, pp. 108 - 115, doi:10.1109/SIITME63973.2024.10814825.

[22] T. Edamatsu and D. Takahashi, "Efficient Large Integer Multiplication with Arm SVE Instructions," in Proc. Int. Conf. High Performance Comput. Asia-Pacific Region (HPC ASIA), Singapore, pp. 1-9, Feb. 2023, doi:10.1145/3578178.3578193.

[23] V. Titopoulos, K. Alexandridis, C. Peltekis, C. Nicopoulos, and G. Dimitrakopoulos, "Optimizing Structured-Sparse Matrix Multiplication in RISC-V Vector Processors," IEEE Trans. Comput., vol. 74, no. 2, pp. 215-228, 2025, doi:10.1109/TC.2025.3533083.

[24] M. Böther, L. Benson, A. Klimovic, and T. Rabl, "Analyzing Vectorized Hash Tables Across CPU Architectures," Proc. VLDB Endowment, vol. 16, no. 11, pp. 2755-2768, 2023, doi:10.14778/3611479.3611485. 24

[25] K. Yang and J. F. Martínez, "VersaTile: Flexible Tiled Architectures via Associative Processors," ACM Trans. Arch. Code Optim., vol. 20, no. 2, pp. 1-28, 2025, doi:10.1145/3716873.