AraXL: A Physically Scalable, Ultra-Wide RISC-V Vector Processor Design for Fast and Efficient Computation on Long Vectors

Navaneeth Kunhi Purayil[*1], Matteo Perotti[*1], Tim Fischer[1] and Luca Benini[1,2]
[1]*ETH Zürich, Zürich, Switzerland*, [2]*Università di Bologna, Bologna, Italy*
{nkunhi,mperotti,fischeti,lbenini}@iis.ee.ethz.ch

***Abstract*—The ever-growing scale of data parallelism in today's HPC and ML applications presents a big challenge for computing architectures' energy efficiency and performance. Vector processors address the scale-up challenge by decoupling Vector Register File (VRF) and datapath widths, allowing the VRF to host long vectors and increase register-stored data reuse while reducing the relative cost of instruction fetch and decode. However, even the largest vector processor designs today struggle to scale to more than 8 vector lanes with double-precision Floating Point Units (FPUs) and 256 64-bit elements per vector register. This limitation is induced by difficulties in the physical implementation, which becomes wire-dominated and inefficient.**

In this work, we present AraXL, a modular and scalable 64-bit RISC-V V vector architecture targeting long-vector applications for HPC and ML. AraXL addresses the physical scalability challenges of state-of-the-art vector processors with a distributed and hierarchical interconnect, supporting up to 64 parallel vector lanes and reaching the maximum Vector Register File size of 64 Kibit/vreg permitted by the RISC-V V 1.0 ISA specification. Implemented in a 22-nm technology node, our 64-lane AraXL achieves a performance peak of 146 GFLOPs on computation-intensive HPC/ML kernels (>99% FPU utilization) and energy efficiency of 40.1 GFLOPs/W (1.15 GHz, TT, 0.8V), with only 3.8× the area of a 16-lane instance.

***Index Terms*—Vector processors, RISC-V, Scalability**

I. Introduction

The amount of data and the computational needs of today's applications have skyrocketed, with no signs of slowing down. This unprecedented growth requires innovative solutions in hardware and software, as technology scaling alone no longer provides a reliable way of boosting chips' performance. Moreover, simply chasing performance through frequency improvements is no longer viable due to the power wall [1]. As a result, prioritizing energy efficiency over sheer performance has become imperative in modern hardware designs, even in the High Performance Computing (HPC) domain [2].

To address this challenge, one of the most efficient solutions is leveraging applications' Data-Level Parallelism (DLP) to encode multiple operations in a single instruction, amortizing its fetch/decode cost, as many crucial HPC and Machine Learning (ML) applications exhibit high degrees of parallelism and often require computation on extremely long vectors.

Single Instruction Multiple Data (SIMD) architectures are able to exploit long vector lengths by simultaneously computing multiple vector elements with a single instruction. However, these architectures are limited by their Vector Register File (VRF) width (i.e., the number of vector elements that can be buffered in the architecture), which is usually the datapath width, hindering the data reuse and the amortization of instruction fetch and decode.

On the other hand, Cray-inspired [3] vector processor architectures feature VRFs whose size is decoupled from the datapath width. The vector length can be programmed at runtime and is upper-bounded by the physical size of each vector register, which is a design parameter. Supporting large vector lengths not only minimizes the energy spent in fetching, decoding, and issuing instructions but also has critical performance benefits. Longer vectors lower pressure on the data memory due to higher data reuse close to the Floating Point Units (FPUs). Further, they exhibit a higher tolerance for stalls and memory latency, resulting in improved performance for both dense [4] and sparse workloads [5]–[8].

Due to these reasons, vector processor architectures are gaining traction. For instance, Arm developed the Scalable Vector Extension (SVE) (2016) and SVE2 (2019) scalable vector Instruction Set Architecture (ISA) extensions, used in the Arm Neoverse V2 and the Fujitsu A64FX cores to power the AWS GRAVITON4 [9] and supercomputer FUGAKU [10], respectively. The open-source RISC-V ISA has also recently ratified its vector extension V 1.0, with a plethora of novel architectures designed by universities [11]–[13] and companies [14]–[16]. RISC-V V, more than Arm SVE, highlights the importance of long vectors, allowing a maximum vector length of 64 Kibit per vector register. However, as of today, no RISC-V vector processor architecture has ever implemented such large VRF.

Scaling up the VRF and the number of parallel FPUs of a vector processor architecture presents numerous challenges. Vitruvius+ [12] and Ara2 [13] are the largest RISC-V V vector processor architectures available and feature up to 8 and 16 parallel lanes with one double-precision FPU per lane, respectively, and a large VRFs with up to 16 Kibit of vector length. Their VRF is split among the vector lanes to improve data locality and limit the interconnect complexity. Despite being a modular lane-based architecture, Ara2 showed that scaling to more than 8 lanes is challenging due to numerous all-to-all interconnects that enable data movements between

[*] The first two authors contributed equally to this work.

978-3-9826741-0-0/DATE25/© 2025 EDAA

the memory and the lanes (VRF) and among the lanes.

In this work, we present AraXL, which leverages long vectors in HPC and ML applications to tolerate memory latency and therefore ease the physical implementation. AraXL solves the scalability challenge of the all-to-all interconnects with a dedicated hierarchical and pipelined interconnect, allowing it to scale up to 64 double-precision floating-point-capable vector lanes.

The key contributions of this paper are:

- AraXL, the first RISC-V vector processor architecture able to scale up to 64 lanes supporting vectors of up to 8192 double-precision (DP)-elements. AraXL tolerates memory latency and overcomes today's vector processor scalability limitations, achieving the longest vector length permitted by the RISC-V V ISA specifications.
- AraXL's physical implementation in a 22-nm technology node and a study of its power, performance, and area (PPA) metrics. AraXL's modular architecture scales up from 2 to 64 lanes with almost perfect area scaling (2× when doubling the lane count and VRF size). The maximum frequency is never lower than 1.15 GHz with an energy efficiency of 40 GFLOPs/W (TT, 0.8V, 25C).
- An evaluation of AraXL's performance and tolerance to memory latency on multiple compute- and memory-intensive kernels, showing almost perfect performance scaling under weak scaling conditions. AraXL with 64 lanes can reach up to 99% utilization on the `fmatmul` matrix multiplication kernel.

II. Related works

Significant progress has been made in the development of scalable and energy-efficient vector processors, particularly following the introduction of the RISC-V V vector extension. Figure 1 summarizes the most notable ones, shown by their vector register length and number of processing units (FPUs).

Many of these processors are designed for applications that do not exhibit extremely high vector lengths and typically feature a limited VRF and fewer FPUs. These designs often focus on smaller workloads and leverage multicore configurations to exploit Thread-Level Parallelism (TLP) or other dimensions of parallelization. Examples of such processors include most of the SiFive vector architectures [14], [17], [18], Spatz [19], small instances of Ara2/Vicuna [11], [13], and Arrow [20]. These architectures are not primarily aimed at HPC applications, where long vectors are exposed, and high FPU counts become crucial.

Vector processors targeting the HPC domain also exist and typically feature a higher number of FPUs. For instance, the Fujitsu A64FX is made of four Core Memory Groups (CMGs), each composed of 12 computing cores. Each core has 2 FPUs delivering a total of 32 double-precision operations per cycle [10]. However, A64FX's cores implement Arm SVE, which limits the vector register length to 2048 bits, preventing aggressive exploitation of the benefits of long vectors. Some RISC-V vector processors with multiple FPUs also have limited register file capacity: the configurable vector units from Andes

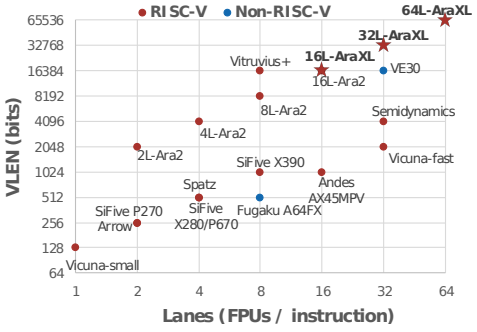

Fig. 1. Vector processors grouped by vector register bit-width (VLEN) and number of FPU used by a vector instruction.

AX45MPV [21] and Semidynamics [22] feature VRFs up to 16 and 32 FPUs, respectively, but with VLEN (bit-width of a vector register) limited to 1024 and 4096 bit.

Vector processors targeting long vectors with higher FPU counts exist as well. Notable examples include Vitruvius+, the 8- and 16-lane Ara2 instances, and NEC's TSUBASA Aurora.

Vitruvius+, part of the European Processor Initiative, is tailored for HPC applications that expose long vectors, supporting 8 lanes and a VLEN of 16 Kibits [12]. Ara2 is another RISC-V V processor with a VLEN of up to 16 Kibits and 16 FPUs, but scaling Ara2 microarchitecture beyond 8 lanes is challenging due to the complexity of all-to-all interconnects that allow data movements among the lanes (in the load-store, mask, and slide units) [13].

Today's largest-scale vector processor is not a RISC-V one, though. NEC's TSUBASA Aurora explicitly targets extremely long vectors with its multi-core VE30 vector engines [23]. A single core features 32 lanes, each with 3 FMAs, 2 ALUs, and a Complex/Store pipeline, totaling 6 execution pipelines connected to a multi-ported VRF with a VLEN of 16 Kibits. However, despite its theoretical peak performance, its ability to buffer elements in the register file is limited compared to the RISC-V V specifications. Moreover, the microarchitecture of the VE30 is proprietary, and effective performance and power efficiency have not been benchmarked in the open literature, making it unclear to what extent their performance and efficiency meet theoretical peaks in practical workloads.

Our architecture, AraXL, is designed to tackle the scalability challenges observed in current high-VLEN RISC-V vector processors by means of implementation-friendly and modular interconnects. This allows AraXL to feature 64 FPUs and the highest VLEN allowed by RISC-V V (64Kibit per register), maximizing the latency tolerance and power benefits from HPC and ML long-vector applications. In fact, higher VLENs up to 64 Kibit can increase the HPC-workload performance [8] and allow leveraging context windows as large as 128k elements in Llama3 [24] in the ML domain.

III. Architecture

A. Architecture overview

AraXL is a decoupled vector processor architecture composed of a CVA6 scalar core [25] and multiple physical

clusters that act as a single RISC-V V accelerator.

As depicted in Figure 2, AraXL's hierarchical architecture is based on vector clusters, each composed of an enhanced instance of the open-source Ara2 [13] equipped with its own dispatcher, sequencer, all-to-all (A2A)-interconnected units (Mask Unit (MASKU), Slide Unit (SLDU) and Vector Load Store Unit (VLSU)), and lanes, which feature the processing units and the Vector Register File (VRF) chunks. We choose the 4-lane configuration as a building block for the vector cluster, as it features the highest energy efficiency [13] among all the configurations, and we modify it by streamlining its internal interconnects.

We design three scalable interfaces - the Request Interface (REQI), Global Load Store unit (GLSU) interface, and Ring Interface (RINGI) - to connect the clusters to the CVA6 scalar core, the L2 memory, and the neighbor clusters for permutation operations, respectively.

Conceptually, AraXL is a set of vector processor clusters synchronized through the REQI, operating on vector elements mapped from memory to the internal sparse VRF by the GLSU, and using the RINGI to move data among different clusters. To achieve a scalable architecture, we prioritize relaxing the timing of all top-level interconnects over their latency, which is not critical in long-vector applications. This also means that all the possibly-critical paths through the interfaces can be cut with a parametric number of registers, as shown in Section IV.

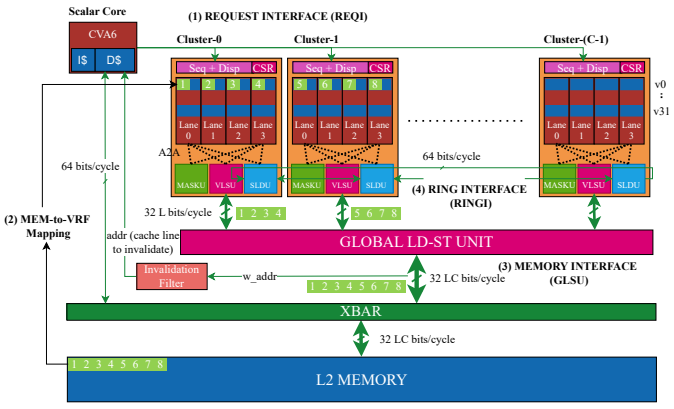

Fig. 2. Schematic of the cluster-based AraXL architecture and its interfaces.

We design AraXL to target maximum performance on the most common subset of RISC-V Vector Extension (RVV) instructions, namely unit-stride memory accesses, slide-by-1 operations, reductions, and basic mask operations. These instructions are the most prevalent in regular data parallel workloads in the HPC and ML domain. Strided/Indexed memory operations, variable slides, and other irregular RVV operations are supported, albeit at lower throughput.

In the following sections, we discuss AraXL's architecture.

B. Architecture details

1) REQI: CVA6 broadcasts the vector instructions fetched by its L1 i-cache to all clusters through the REQI. Then, the clusters decode and execute every instruction in sync.

Once the request is accepted by clusters, cluster-0 sends the answer back to CVA6, signaling possible exceptions or forwarding scalar results to be written to scalar registers.

2) Memory to VRF byte mapping: Ara2 implements an element-wise mapping of memory bytes to lanes, such that element-i always maps to lane-$i(mod)L$, regardless of the Element Width (EW). This ensures that common mixed-width operations, e.g., when accumulations are done in higher precision, do not require scrambling of bytes among lanes [13]. AraXL naturally extends Ara2's byte mapping such that element-i maps to cluster-$i/L(mod)C$, lane-$i(mod)L$. Figure 2 shows AraXL's memory-VRF byte mapping, which happens in two stages: 1) from the memory to the clusters through the GLSU, and 2) from each cluster to its lanes via the local VLSU within the cluster.

3) GLSU: The GLSU receives requests from the local VLSUs and generates a wider Advanced eXtensible Interface (AXI) request to the memory. In Ara2, the byte mapping interconnect of the VLSU showed limited scalability since $8L$ bytes coming from memory are A2A interconnected to each $8L$ byte of every lane to support unaligned load-store AXI requests, resulting in quadratic complexity of L^2 for the VLSU [13].

To achieve a scalable GLSU, AraXL implements the shuffling and aligning logic in a multi-level pipeline to move the memory bytes to the correct clusters in multiple cycles. By integrating this pipelined logic into the interconnect, we trade off latency with higher scalability, which is enabled by the latency tolerance of our target applications.

Figure 3 shows the 3-stage architecture of the GLSU. The *Align* stage aligns the misaligned data to the memory bus with multiple power-of-2 shifts. The *Addrgen* stage handles request splitting and bandwidth conversion. The *Shuffle* stage shuffles the aligned data to different clusters based on the EW configuration. Each level of the *Align* and *Shuffle* stages is guarded by registers and receives control signals based on the address, vector length, and element width, which are tracked in the shuffle and align tables.

As a consequence, the VLSUs local to the clusters only need to shuffle the data bytes to the lanes since aligning has already been done by the GLSU. In contrast, the original VLSU of Ara2 aligns and shuffles the memory bytes to the lane's VRFs in a single cycle, leading to scalability challenges.

4) RINGI: The original Ara2 design uses a lumped SLDU for permutation operations between lanes during, for example, vector slides and reductions.

In AraXL, we implement a ring interconnect to move data among clusters and extend each cluster's SLDU to utilize the data from the ring whenever necessary. We choose the ring interconnect since most HPC and ML workloads utilize slide-by-1 operations requiring only data movement between adjacent clusters. Furthermore, the ring interconnect is easily scalable to a large number of vector clusters.

To maximize the performance of the two most common operations (slide1up, slide1down), each cluster supports two

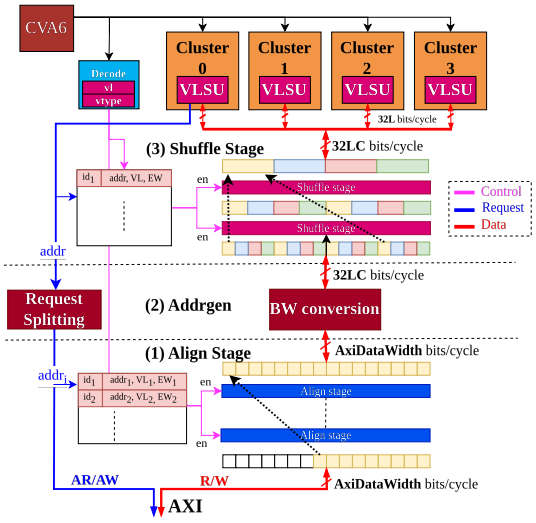

Fig. 3. Schematic of the Global Load-Store Unit (GLSU).

output data busses with a bandwidth of 64 bits/cycle, targeting the previous and the next cluster, together with the two incoming ones. Slides larger than 1 are implemented using multiple 64-bit data transfers or bypasses on the ring to the correct destination lane. A schematic of the `vfslide1down` operation using the ring is shown in figure 4.

The original Ara2 implements three stages for reductions: intra-lane, inter-lane, and a SIMD stage. In AraXL, we add an inter-cluster stage to reduce the values local to each cluster after the inter-lane step, for which we use the ring interconnect. This reduction is done in a log-tree fashion and utilizes multiple hops for later reduction stages. To ensure timing is not affected, AraXL instantiates a parametric number of registers in the ring interconnect between SLDUs of adjacent clusters.

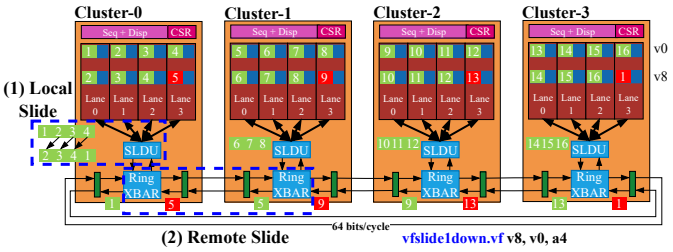

Fig. 4. Schematic of the Ring Interface (RINGI).

5) MASKU: The original Ara2's MASKU contributed to the scalability issues due to its A2A nature at the bit-level to distribute mask bits across different lanes. This would be even more problematic in AraXL, where a 64-lane architecture would require distributing each bit of a 64-bit packet to a different lane.

To prevent this, we add a new dedicated VRF byte encoding to keep the mask vector bits already in the corresponding lane.

The addition of a new byte layout requires supporting reshuffling conversions between this format and the other

Benchmarks	Problem size* [DP Elements]	LMUL	Max Perf [DP-FLOP/cycle]
fmatmul	A=64×256 B=256×N	1,2,4	2×LC
fconv2d	A=256×N f=7×7	2	2×LC
jacobi2d	A=256×N	4	LC
fdotproduct	A=B=N	8	LC
exp	A=N	1	28/21×LC
softmax	A=64×N	1	32/25×LC

* N = nLC (for L lanes per cluster, C clusters, and $n = 16\times$ LMUL)
* Assuming an L2 memory size of at least 16 MiB to fit the benchmarks

TABLE I
BENCHMARK PARAMETERS USED FOR EVALUATIONS.

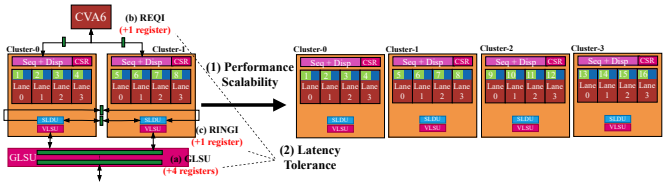

Fig. 5. *Performance scalability* and *Latency tolerance* experiment setups.

supported byte encodings. In AraXL, this is done by the SLDU through the RINGI to move bits across clusters.

As noted for Ara2, reshuffling is a slow operation, and software should not use the same register to sequentially hold mask and non-mask vector elements to avoid unnecessary byte layout modifications.

IV. Evaluation

A. Evaluation setup

We implement AraXL in SystemVerilog and characterize its performance with configurations up to 64 lanes using cycle-accurate simulations with QUESTASIM-2021.2. To benchmark AraXL, we use a selection of common HPC/ML kernels whose instructions include unit-stride load-stores, slide-by-1 (`fconv2d`, `jacobi2d`), reduction (`fdotproduct`, `softmax`) and basic mask operations (`exp`) (Table I). We evaluate AraXL's performance with two metrics: *performance scalability*, with a comparison against the original Ara2, and *latency tolerance* (Figure 5).

Finally, we synthesize and place-and-route AraXL in 22-nm technology with SYNOPSYS DC and IC COMPILER 2 2022.03 and discuss its PPA metrics and scalability for up to 64 lane configurations. We extract the power consumption of the post-layout netlist with power simulations using PRIME-TIME 2022.03 in the typical conditions (0.8V, TT, 25C).

B. Performance Scalability

We characterize AraXL's performance scalability under weak scaling conditions by simulating our kernels on larger AraXL configurations at proportionally larger problem sizes, ensuring that each lane always operates on the same number of bytes. Figure 6 reports the measured performance values normalized to the original 8-lane Ara2's performance up to 512

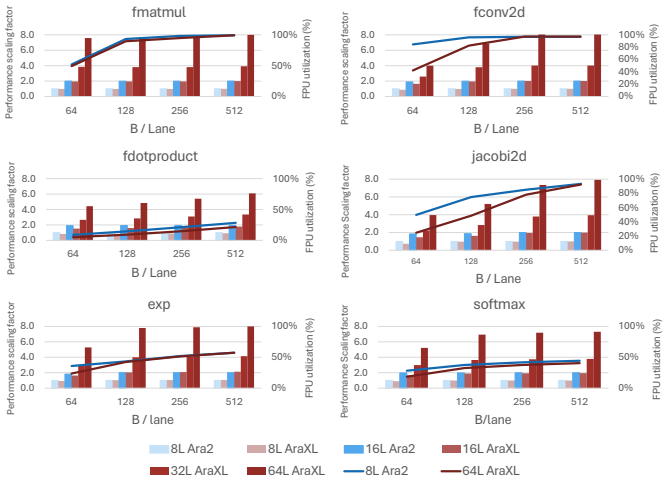

Fig. 6. AraXL's performance scalability. The bars (left Y-axis) are normalized on the original 8-lane Ara2's performance. The lines (right Y-axis) represent absolute FPU utilization.

B/lane (bar plot, left Y-axis). The FPU utilization, measured as the percentage of runtime in which the FPU is producing valid results, is also reported for the 64-lane AraXL and 8-lane Ara2 (lines, right Y-axis).

In the medium vector length regime (64 B/lane), both the original Ara2 design and AraXL show lower FPU utilization since they cannot hide the latency determined by the setup time of the vector instructions and the latency of scalar loads-stores through the data-cache. This effect is worse in AraXL since the newly designed interfaces increase the vector instruction setup time.

In the long vector regime (from 128 B/lane), which is the explicit target of AraXL, the vector pipeline is busy enough to hide the scalar setup time and interface latencies, leading to high FPU utilization on the computationally intensive kernels. As can be seen from the figure, `fmatmul` and `fconv2d` achieve up to 99% and 97% utilization, respectively, and linear performance scaling from 8 to 64 lanes. This trend is similar for the other two compute-bound `exp` and `jacobi2d` kernels. On the other hand, `softmax` and the memory-bound `fdotproduct` kernels use reduction operations, which incurs a noticeable FPU utilization drop even for longer vectors, with performance scaling factors of 7.3× and 6.1× on a 64-lane AraXL instance, respectively. This slight trend degradation is caused by the non-ideal inter-lane log-tree reduction steps funneled through the ring interconnect. Since the inter-cluster and inter-lane reduction latencies depend on the architecture's configuration and not on the problem size, an even larger vector length mitigates these non-idealities. For example, AraXL can achieve a close-to-linear performance scaling of 7.6× with a 16384 B/lane vector dot product, strip-mined over 16 loop iterations, as the time spent to partially reduce the vector elements locally to each lane (intra-lane phase) amply dominates the total reduction time, amortizing the non-ideal inter-lane and inter-cluster steps.

Overall, we conclude that AraXL achieves linear performance scaling from 8 to 64 lanes when processing long-vector workloads for all the benchmarks, with high FPU utilization, especially for the crucial `fmatmul` and `fconv2d` kernels.

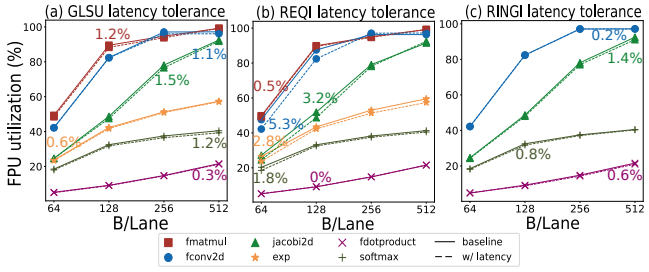

Fig. 7. Performance impact of additional latency on the (a) Memory, (b) Request, and (c) Ring interfaces over the 64-lane AraXL baseline. For each kernel, the max. FPU utilization drop is reported.

C. Latency tolerance

We also evaluate the impact of the additional latency caused by the insertion of sequential cuts into the cluster interfaces as depicted in Figure 5. Figure 7 shows the latency tolerance of AraXL in terms of FPU utilization degradation with the addition of 4, 1, and 1 register cuts to the GLSU, REQI, and RINGI interfaces, respectively.

a) GLSU interface: The register additions along the GLSU interface increase the request-response latency by 8 cycles. As shown in Figure 7 (a), the maximum utilization drop in the long-vector regime is a mere 1.5%. Furthermore, longer vectors face virtually no performance drop.

b) REQI: Adding a register on the REQI implies that the vector instruction is acknowledged back to CVA6 2 cycles later, delaying the issue of the next instruction. From Figure 7 (b), we see a maximum utilization drop of 5% for `fconv2d` and 3% for `jacobi2d` at 128 B/lane. However, this can be completely amortized at 512 B/lane for both kernels.

c) RINGI: The added registers increase the 1-hop latency between clusters by 1 cycle, which affects slide and reduction operations. However, Figure 7 (c) shows that, for long vectors, we only see up to 1.4% drop in utilization.

Overall, AraXL exhibits high latency tolerance on all three interfaces - GLSU, REQI and RINGI - achieving less than 2% utilization drop in the long-vector regime.

D. Physical Implementation

We perform a hierarchical physical implementation of AraXL in a 22-nm technology with configurations of 16, 32, and 64 lanes and evaluate its Power Performance Area (PPA) metrics and scalability. We show the annotated 16-lane AraXL floorplan in Figure 8.

a) Area and Timing: In Figure 9, we compare the area of a 16-lane AraXL against the original 16-lane Ara2 architecture. AraXL achieves a significant area improvement of 14% from the redesign of the A2A units - MASKU, SLDU, and VLSU - which limited the scalability of Ara2 beyond 8 lanes. AraXL also reaches a higher maximum frequency of 1.4 GHz, an improvement from 1.08 GHz in typical conditions (0.8V,

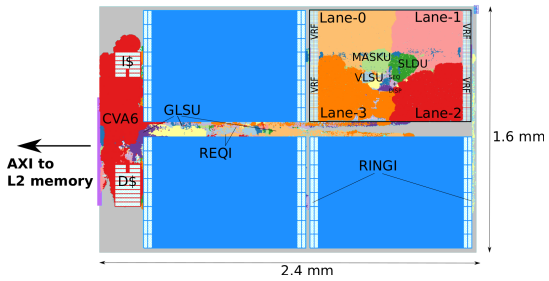

Fig. 8. 16-lane AraXL floorplan. Focus on the modified 4-lane Ara2 used as a cluster, its A2A connected units, CVA6, and the top-level interfaces.

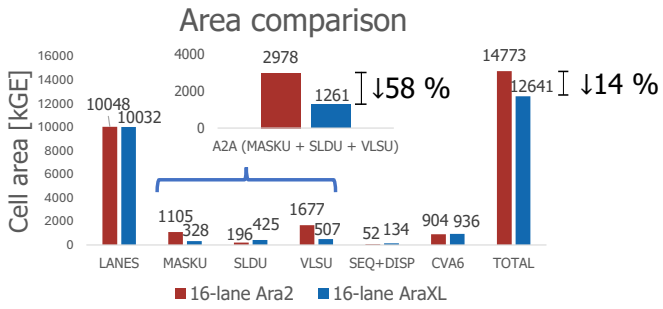

Fig. 9. Area breakdown of 16-lane AraXL and Ara2. For a fair comparison, AraXL's VLSU, SLDU, and SEQ+DISP also include the area of the top-level GLSU, RINGI, and REQI interfaces.

TT, 25C). This is enabled by the memory latency tolerance of AraXL shown in Section IV-C, which allows removing Ara2's critical path from the align/shuffle complexity of the A2A connected VLSU and MASKU.

Table II shows the area scaling trends of AraXL from 16 to 64 lanes. We see that both 32- and 64-lane AraXL achieve linear scaling w.r.t 16-lane AraXL thanks to the cost-effective interconnect design. The GLSU, RINGI, and REQI account for only 3% of the total area.

AraXL reaches 1.4 GHz up to 32 lanes, with a frequency degradation for the 64-lane design (1.15 GHz) due to floorplan inefficiencies that result in routing congestion hotspots.

b) Efficiency comparison: We calculate the energy efficiency metrics for 16-, 32-, and 64-lane AraXL simulating `fmatmul` in the long-vector regime (512 B/lane) in the typical conditions (0.8V, TT, 25C). As shown in Table III, AraXL achieves an energy efficiency of 40.4 GFLOPS/W and an area efficiency of 17.8 GFLOPS/mm^2 showing significant improvements w.r.t 16-lane Ara2.

Cell area [kGE] (×*)	16L-AraXL	32L-AraXL	64L-AraXL
Clusters	11354 (1.0×)	22708 (2.0×)	45415 (2.0×)
CVA6	936 (1.0×)	901 (1.0×)	931 (1.0×)
GLSU	291 (1.0×)	618 (2.1×)	1385 (2.2×)
RINGI	25 (1.0×)	44 (1.8×)	76 (1.7×)
REQI	34 (1.0×)	81 (2.4×)	144 (1.8×)
TOTAL	12641 (1.0×)	24352 (1.9×)	47950 (2.0×)

* Scaling factor normalized to half the number of lanes

TABLE II

ARAXL AREA BREAKDOWN AND SCALING CHARACTERIZATION.

	L	Freq.* [GHz]	Max. Perf. [GFLOPs]	Energy Eff. [$\frac{\text{GFLOPs}}{\text{W}}$]	Area Eff. [$\frac{\text{GFLOPs}}{\text{mm}^2}$]
Vitruvius+	8	1.40	22.4	47.3**	17.23**
Ara2	16	1.08	34.2	30.3	11.6
AraXL	16	1.40	44.3	39.6	17.4
AraXL	32	1.40	87.2	40.4	17.8
AraXL	64	1.15	146.0	40.1	15.1

* Typical corner max freq. ** Scalar core and caches not included.

TABLE III

ARAXL PPA COMPARISON AGAINST SOA LANED VECTOR PROCESSORS.

E. SoA comparison

AraXL is the first RVV 1.0 vector processor architecture to feature 64 FPUs and a VLEN of 64 Kibits (2× and 4× compared to the largest count reported so far, respectively). AraXL architecture improves Ara2's energy and area efficiencies by 30% and 50%, respectively, for the same number of lanes, with a +30% maximum frequency and no FPU utilization drop in the long-vector regime. Our 32-lane configuration achieves 4× the performance of Vitruvius+ with similar area efficiency and the same frequency. Comparing the energy efficiency is harder since the scalar core and cache power consumptions are not included in Vitruvius+'s metric.

A PPA comparison with the VE30 vector engine is not straightforward since only total system power (including interconnects and caches) and area numbers are reported in the literature, making it hard to perform a standalone comparison of the vector unit. Nevertheless, compared to the area evaluations performed in [12], AraXL reaches at least +45% better area efficiency than the older-generation VE NEC vector unit (10.16 DP-GFLOPS/mm^2 at 1.6 GHz).

V. CONCLUSIONS

In this work, we presented AraXL, a novel RISC-V vector architecture designed to leverage long-vector applications in the HPC and ML domains. AraXL features the maximum VRF size allowed by the V 1.0 specifications (64 Kibit/register) and can scale up to 64 parallel vector lanes with linear scaling (2× the area with twice the lanes) thanks to dedicated optimizations to the all-to-all interconnects that usually limit the scalability of today's vector processors.

We implement AraXL in an advanced 22-nm technology node reaching 1.15 GHz and an efficiency of 40.1 GFLOPS/W (0.8V, TT, 25C) for a 64-lane configuration. AraXL's performance on multiple compute- and memory-intensive kernels doubles when doubling the number of lanes with long vectors in weak-scaling conditions, reaching more than 99% utilization on sufficiently large matrix multiplications even with 64 lanes.

VI. ACKNOWLEDGMENTS

This project was supported in part through the ISOLDE (101112274) project that received funding from the HORIZON CHIPS-JU programme and in part from the Swiss State Secretariat for Education, Research, and Innovation (SERI) under the SwissChips initiative.

REFERENCES

[1] O. Villa *et al.*, "Scaling the power wall: A path to exascale," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014.

[2] E. Masciari and E. V. Napolitano, "The environmental cost of high performance computing system simulation," in *32nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2024.

[3] R. Russell, "The CRAY-1 computer system," *Communications of ACM*, vol. 21, no. 1, pp. 63–72, Jan. 1978.

[4] C. Ramírez *et al.*, "A RISC-V Simulator and Benchmark Suite for Designing and Evaluating Vector Architectures," *ACM Transactions on Architecture and Code Optimization*, vol. 17, no. 4, pp. 38:1–38:30, Nov. 2020.

[5] S. R. Gupta, N. Papadopoulou, and M. Pericàs, "Challenges and opportunities in the co-design of convolutions and RISC-V vector processors," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. ACM, 2023.

[6] P. Vizcaino *et al.*, "Short Reasons for Long Vectors in HPC CPUs: A Study Based on RISC-V," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. ACM, 2023.

[7] C. Gómez, F. Mantovani, E. Focht, and M. Casas, "Efficiently running SpMV on long vector architectures," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2021.

[8] ——, "HPCG on long-vector architectures: Evaluation and optimization on NEC SX-Aurora and RISC-V," *Future Generation Computer Systems*, vol. 143, pp. 152–162, Jun. 2023.

[9] A. W. Services, "AWS Graviton - Getting started," accessed April 28, 2024. [Online]. Available: https://github.com/aws/aws-graviton-getting-started

[10] R. Okazaki *et al.*, "Supercomputer Fugaku CPU A64FX realizing high performance, high-density packaging, and low power consumption," *Fujitsu Technical Review*, 2020. [Online]. Available: https://www.fujitsu.com/global/documents/about/resources/publications/technicalreview/2020-03/article03.pdf

[11] M. Platzer and P. Puschner, "Vicuna: A timing-predictable RISC-V vector coprocessor for scalable parallel computation," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl, 2021.

[12] F. Minervini *et al.*, "Vitruvius+: An area-efficient RISC-V decoupled vector coprocessor for high performance computing applications," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 2, pp. 1–25, 2023.

[13] M. Perotti *et al.*, "Ara2: Exploring single- and multi-core vector processing with an efficient RVV 1.0 compliant open-source processor," *IEEE Transactions on Computers*, vol. 73, no. 7, pp. 1822–1836, 2024.

[14] *SiFive Intelligence X280*, SiFive Corp., 2022, accessed on January 13, 2025. [Online]. Available: https://www.sifive.com/document-file/x280-datasheet

[15] SiFive, "P870 high-performance RISC-V processor," in *Hot Chips: A Symposium on High-Perf. Chips*. IEEE, 2023.

[16] "AndesCore™ NX27V Processor," Andes Technology, accessed March 31, 2024. [Online]. Available: http://www.andestech.com/en/products-solutions/andescore-processors/riscv-nx27v

[17] SiFive, "SiFive announces differentiated solutions for generative AI and ML applications leading RISC-V into a new era of high-performance innovation," accessed March 31, 2024. [Online]. Available: https://www.sifive.com/press/sifive-announces-differentiated-solutions-for-generative

[18] *SiFive Performance P270*, SiFive Corp., 2022, accessed January 10, 2025. [Online]. Available: https://www.sifive.com/document-file/p270-and-p270-mc-data-sheet

[19] M. Perotti, S. Riedel, M. Cavalcante, and L. Benini, "Spatz: Clustering compact RISC-V-based vector units to maximize computing efficiency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025, Early Access, DOI: 10.1109/TCAD.2025.3528349.

[20] I. A. Assir, M. E. Iskandarani, H. R. A. Sandid, and M. A. R. Saghir, "Arrow: A RISC-V vector accelerator for machine learning inference," in *Fifth Workshop on Computer Architecture Research with RISC-V (CARRV 2021)*, 2021. [Online]. Available: https://carrv.github.io/2021/papers/CARRV2021_paper_100_AlAssir.pdf

[21] "AndesCore™ AX45MPV," Andes Technology, accessed March 31, 2024. [Online]. Available: https://www.andestech.com/en/products-solutions/andescore-processors/riscv-ax45mpv

[22] "Semidynamics vector unit," Semidynamics., accessed March 31, 2024. [Online]. Available: https://semidynamics.com/en/technology/vector-unit

[23] "NEC SX-Aurora TSUBASA architecture," NEC Corporation, accessed March 31, 2024. [Online]. Available: https://www.nec.com/en/global/solutions/hpc/sx/architecture.html

[24] "The Llama 3 herd of models," Meta Platforms Inc., accessed January 10, 2025. [Online]. Available: https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

[25] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-Bit RISC-V core in 22-nm FDSOI technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.