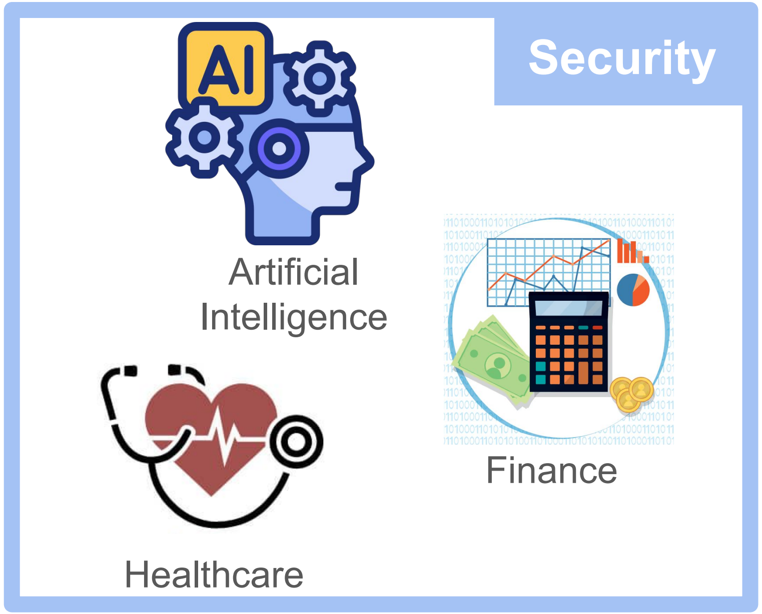
Carnegie Mellon University

A RISC-V Vector Extension for Multi-word Arithmetic

Yunhao Lan, Larry Tang, Naifeng Zhang, Youngjin Eum, James Hoe, Franz Franchetti

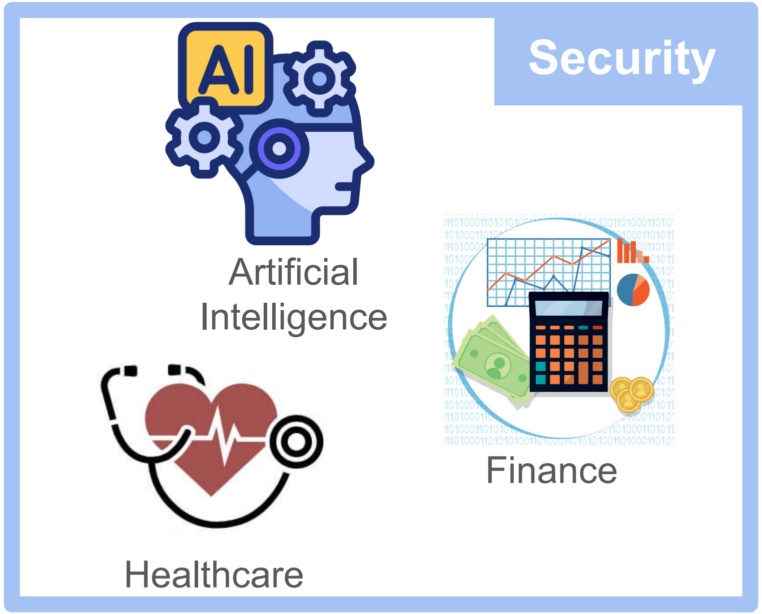
RISCV-HPC
Nov. 17 2025

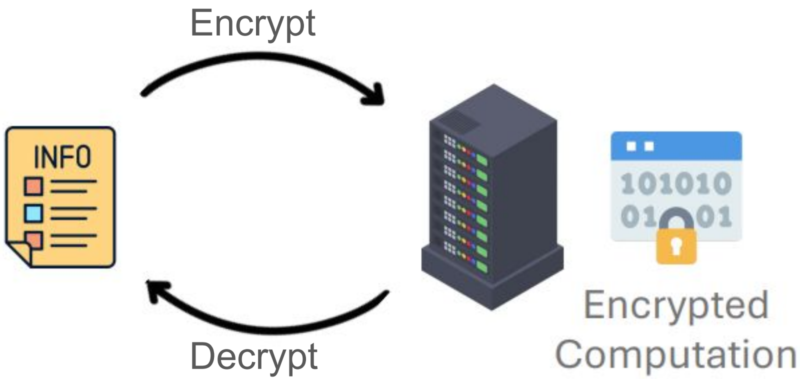
Great Data Security Comes at a High Cost



Zhang, N., Fu, S., & Franchetti, F. (2025). Towards Closing the Performance Gap for Cryptographic Kernels Between CPUs and Specialized Hardware. In Proceedings of MICRO 2025 (IEEE/ACM).

Great Data Security Comes at a High Cost



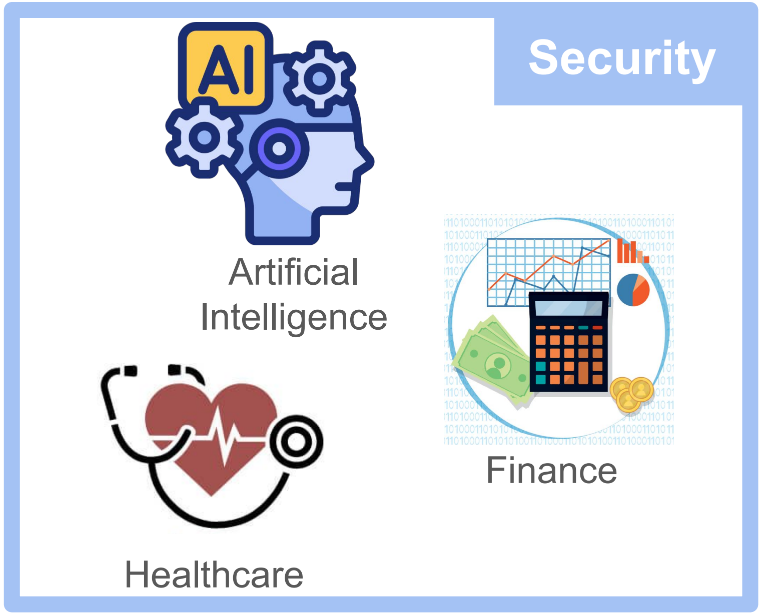


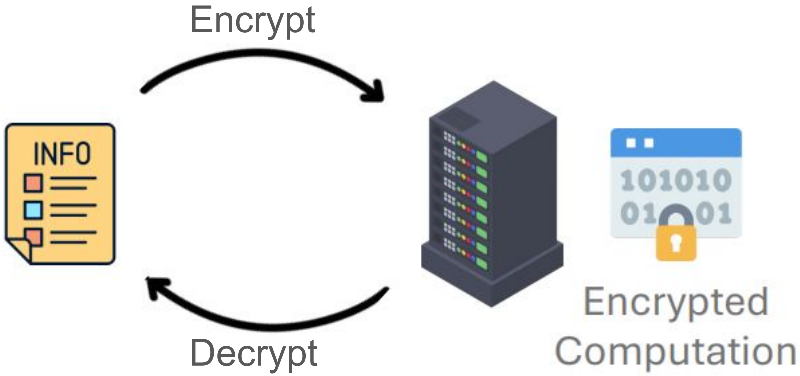
Fully homomorphic encryption (FHE)

Figure borrowed from Naifeng Zhang

Zhang, N., Fu, S., & Franchetti, F. (2025). Towards Closing the Performance Gap for Cryptographic Kernels Between CPUs and Specialized Hardware. In Proceedings of MICRO 2025 (IEEE/ACM).

Great Data Security Comes at a High Cost





Fully homomorphic encryption (FHE)

Figure borrowed from Naifeng Zhang

Cost: Prohibitive overhead dominated by multi-word arithmetic

Zhang, N., Fu, S., & Franchetti, F. (2025). Towards Closing the Performance Gap for Cryptographic Kernels Between CPUs and Specialized Hardware. In Proceedings of MICRO 2025 (IEEE/ACM).

Multi-Word Arithmetic in HPC

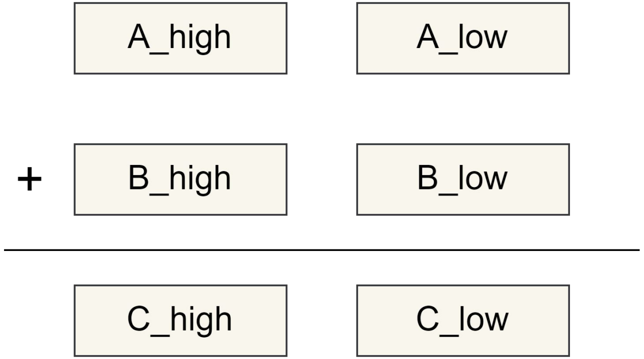


Figure 1: A 128-bit multi-word sum using two 64-bit words.

Multi-Word Arithmetic in HPC

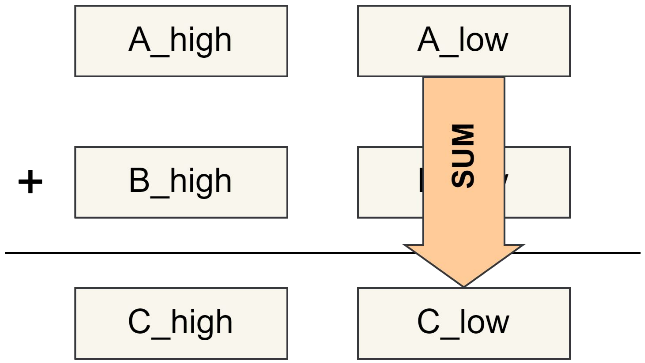


Figure 1: A 128-bit multi-word sum using two 64-bit words.

Multi-Word Arithmetic in HPC

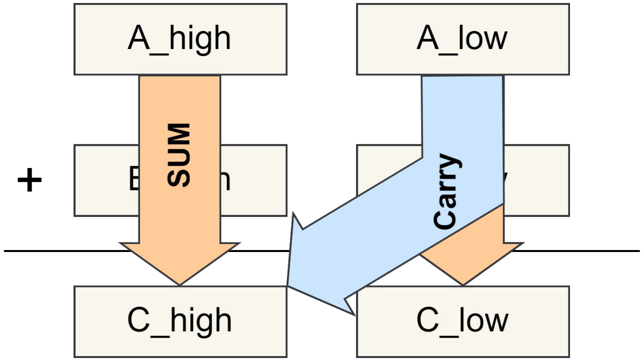


Figure 1: A 128-bit multi-word sum using two 64-bit words.

Multi-Word Arithmetic in HPC

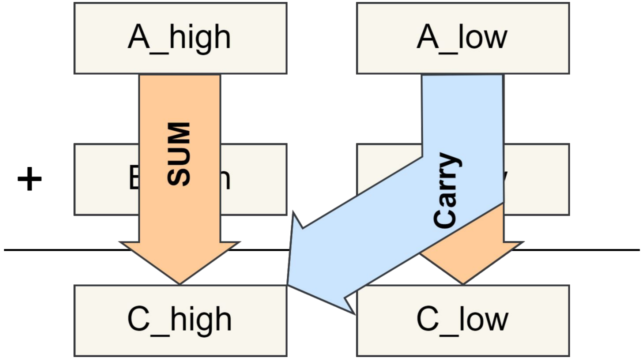


Figure 1: A 128-bit multi-word sum using two 64-bit words.

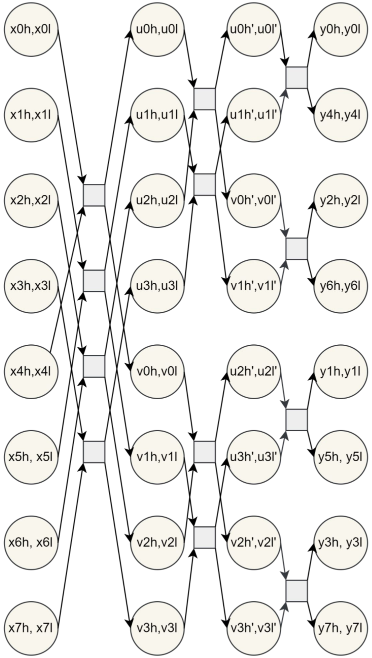


Figure 2: An 8-point 128-bit NTT kernel with multi-word arithmetic on 64-bit systems.

Multi-Word Arithmetic in HPC

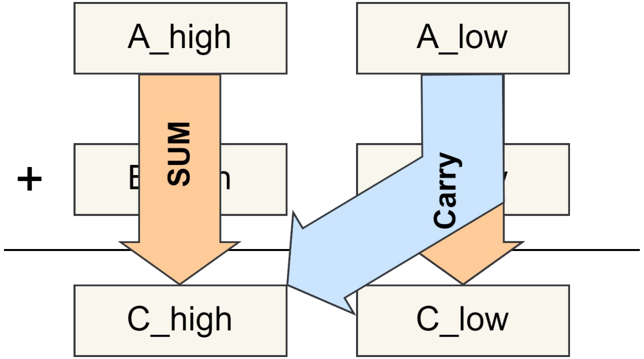


Figure 1: A 128-bit multi-word sum using two 64-bit words.

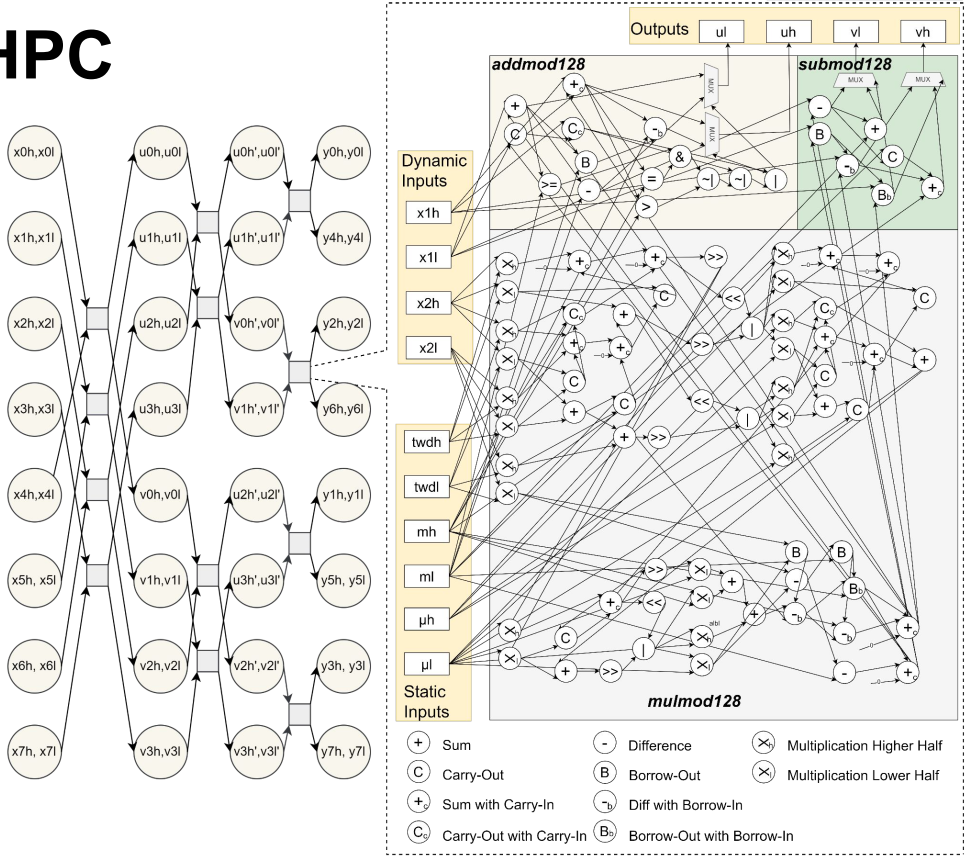


Figure 2: An 8-point 128-bit NTT kernel with multi-word arithmetic on 64-bit systems.

User-Defined Mask Register

```
vmadc.vv v0, v9, v8
vmv1r.v v12, v0
vmadc.vvm v0, v11, v10, v0
vmadc.vvm v15, v14, v13, v0
mand.mm v17, v15, v12
```

In RVV1.0, mask is only provided by v0.

User-Defined Mask Register

```
vmadc.vv v0, v9, v8
vmv1r.v v12, v0
vmadc.vvm v0, v11, v10, v0
vmadc.vvm v15, v14, v13, v0
mand.mm v17, v15, v12
```

```
vmadc.vv v12, v9, v8
vmadc.vvm v0, v11, v10, v12
vmadc.vvm v15, v14, v13, v0
mand.mm v17, v15, v12
```

In RVV1.0, mask is only provided by v0.

Encode the mask register field into the instruction.

CMU

Unified Vector Carry Arithmetic

```
vadd.vv v10, v9, v8   Sum
vmadd.vv v0, v9, v8   Carry

vadc.vvm v13, v12, v11, v0
vmadc.vvm v0, v12, v11, v0
```

In RVV1.0, two instructions are used to compute result and carry separately from the same set of operands.

Unified Vector Carry Arithmetic

```
vadd.vv v10, v9, v8
vmadd.vv v0, v9, v8

vadc.vvm v13, v12, v11, v0
vmadc.vvm v0, v12, v11, v0
```

Sum
Carry

In RVV1.0, two instructions are used to compute result and carry separately from the same operands.

Sum
Carry

```
vadqc.vv v10, v0, v9, v8

vadcqc.vv v13, v0, v12, v11, v0
```

Output result and carry at the same time.

Fused Bitwise Operation and Comparison

From analyzing SPIRAL NTTX-generated kernels, we find the NTT consistently lowers to long dependency chains with two fundamental templates, as follows:

1) Bitwise operation after having paralleled comparison/shifting, e.g., $(a > b)\&(c == d)$;

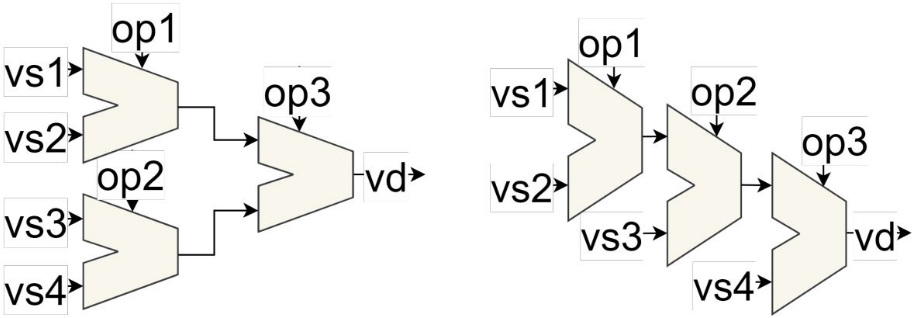
2) Stacked bitwise, e.g., $\sim(c|(\sim(b|a)))$.

Fused Bitwise Operation and Comparison

From analyzing SPIRAL NTTX-generated kernels, we find the NTT consistently lowers to long dependency chains with two fundamental templates, as follows:

1) Bitwise operation after having paralleled comparison/shifting, e.g., $(a > b)\&(c == d)$;

2) Stacked bitwise, e.g., $\sim (c|\,(\sim (b|a)))$.

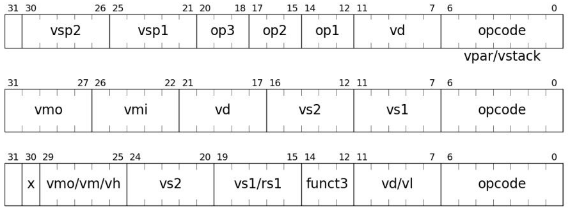


(a) Bitwise Operation after Paralleled Comparison/Shifting. **(b) Stacked Bitwise Operation.**

Proposed ISE and Encoding

Instruction	Description
(A) User-Define Mask Registers	
Vector Load Operation	
`udmvle64.v vd,(rs1),vm`	Vector load for 64-bit elements with user-defined mask-in vm.
Vector–Vector Operations	
`udmvadc.vvm vd, vs1, vs2, vm`	Vector-Vector add with carry; allow user-defined mask-in vm.
`udmvsbc.vvm vd, vs1, vs2, vm`	Vector-Vector subtract with borrow; allow user-defined mask-in vm.
Vector–Scalar Operations	
`udmvadc.vxm vd, vs1, rs2, vm`	Vector-Scalar add with carry; allow user-defined mask-in vm.
`udmvsbc.vxm vd, vs1, rs2, vm`	Vector-Scalar subtract with borrow; allow user-defined mask-in vm.
(B) Unified Vector Carry Arithmetic	
Vector–Vector Operations	
`vadcq.vv vd, vmo, vs1, vs2`	Vector-Vector add; output result vd and carry-out vmo.
`vadcqc.vv vd, vmo, vs1, vs2, vmi`	Vector-Vector add with carry; allow user-defined mask-in vmi; output result vd and carry-out vmo.
`vsbcq.vv vd, vmo, vs1, vs2`	Vector-Vector subtract; output result vd and borrow-out vmo.
`vsbcqc.vv vd, vmo, vs1, vs2, vmi`	Vector-Vector subtract with borrow; allow user-defined mask-in vmi; output result vd and borrow-out vmo.
`vmulf.vv vh, vl, vs1, vs2`	Vector-Vector multiplication; output both lower vl and upper vh parts of the product.
Vector–Scalar Operations	
`vadcq.vx vd, vmo, vs1, rs2`	Vector-Scalar add; output result vd and carry-out vmo.
`vadcqc.vx vd, vmo, vs1, rs2, vmi`	Vector-Scalar add with carry; allow user-defined mask-in vmi; output result vd and carry-out vmo.
`vsbcq.vx vd, vmo, vs1, rs2`	Vector-Scalar subtract; output result vd and borrow-out vmo.
`vsbcqc.vx vd, vmo, vs1, rs2, vmi`	Vector-Scalar subtract with borrow; allow user-defined mask-in vmi; output result vd and borrow-out vmo.
`vmulf.vx vh, vl, vs1, rs2`	Vector-Scalar multiplication; output both lower vl and upper vh parts of the product.
(C) Fused Bitwise Operation and Comparison	
`vpar vd, op[1-3], vsp1, vsp2`	Fused bitwise/comparison with 3 operators (op1, op2, and op3) in parallel pattern.
`vstack vd, op[1-3], vsp1, vsp2`	Fused bitwise/comparison with 3 operators (op1, op2, and op3) in stacked pattern.

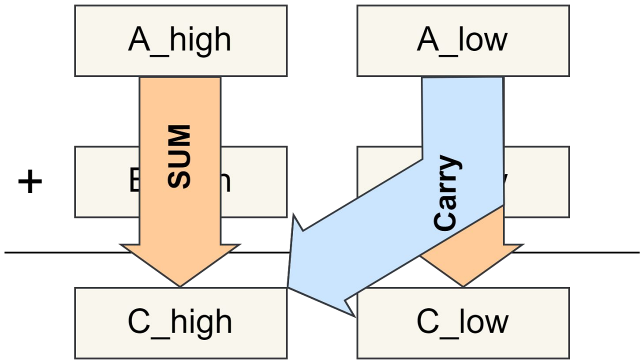
Table 1: Multi-Word extension on RVV.



x	vmo/vm/vh	vs2	vs1/rs1	funct3	vd/vl	opcode
–	–	–	rs1	udmvle64.v	vd	vcustom
vvm/vxm	vm	vs2	vs1	udmvadc	vd	vcustom
vvm/vxm	vm	vs2	vs1	udmvsbc	vd	vcustom
vv/vx	vmo	vs2	vs1	vadcq	vd	vcustom
vv/vx	vmo	vs2	vs1	vsbcq	vd	vcustom
vv/vx	vh	vs2	vs1	vmulf	vl	vcustom

Figure 3: Multi-Word extension instruction encoding.

Example: Double-Word Addition



We intentionally create a **v0 conflict** around this **double-word addition** by inserting mask and load operations.

Example: Double-Word Addition

```
vmor.mm v22,v22,v28

vadd.vv v26,v28,v25
//lower-half addition in v26
vmadc.vv v0,v28,v25

vadc.vvm v25,v27,v24,v0
//higher-half addition in v25
vmadc.vvm v28,v27,v24,v0 //overflow in v28

vmv1r.v v0,v22
vle64.v v24,(a5),v0.t //load with mask of v22
```

Listing 3: Example code snippet with RVV ISA.

Example: Double-Word Addition

```
vmor.mm v22,v22,v28

vadd.vv v26,v28,v25
//lower-half addition in v26
vmadc.vv v0,v28,v25

vadc.vvm v25,v27,v24,v0
//higher-half addition in v25
vmadc.vvm v28,v27,v24,v0 //overflow in v28

vmv1r.v v0,v22
vle64.v v24,(a5),v0.t //load with mask of v22
```

Listing 3: Example code snippet with RVV ISA.

```
vmor.mm v22,v22,v28

vadcq.vv v26,v27,v28,v25
//lower-half addition in v26

vadcqc.vv v25,v27,v28,v25,v27
//higher-half addition in v25; overflow in v28

udmvle64.v v24,(a5),v22
//load with mask of v22
```

Listing 4: Example code snippet with multi-word extension.

Evaluation: Testing Platform

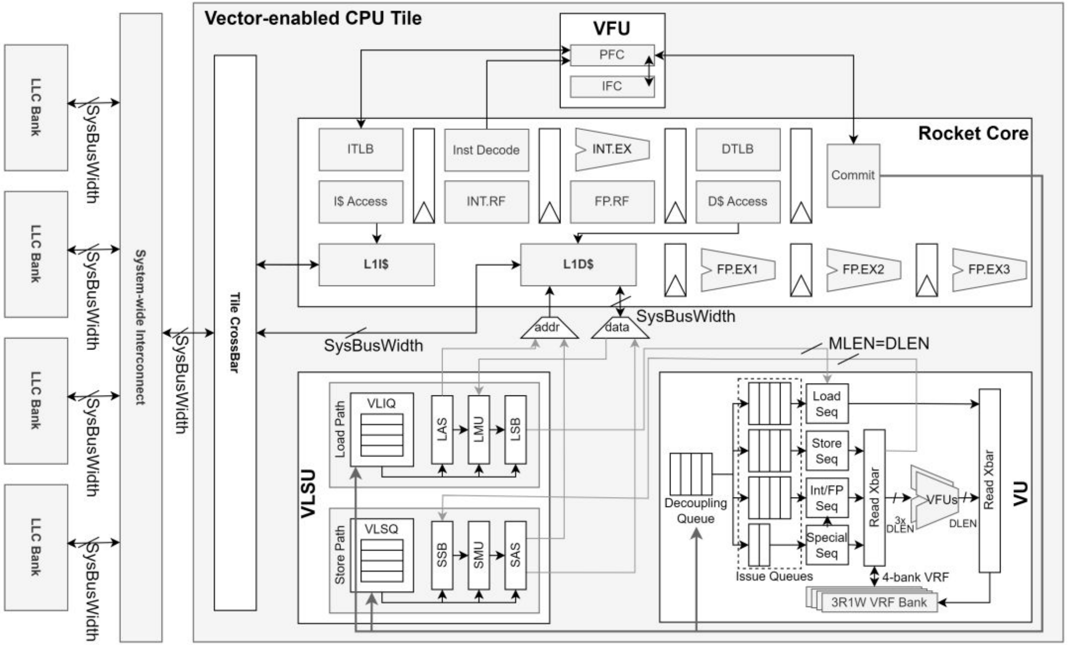


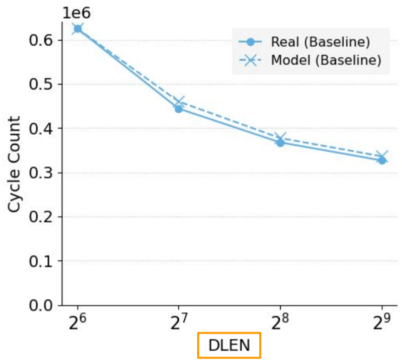
Figure 2: Saturn vector unit integrated on rocket chip architecture diagram.

Performance Models

Performance Models

Datapath Width (DLEN) Model:

$$\varsigma' = \varsigma \cdot (1 - \rho \cdot f + \rho \cdot \frac{f \cdot \delta}{\delta'})$$



, where Cycle (ς), scaled cycle (ς'), VLEN (ν), scaled VLEN (ν'), DLEN (δ), scaled DLEN (δ'), fraction of the parallel compute instruction (f), fraction of parallel compute cycles contributing to the arithmetic (ϱ).

CMU

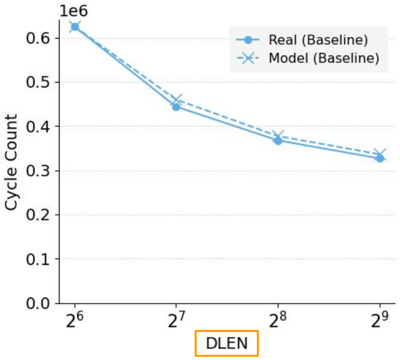
Performance Models

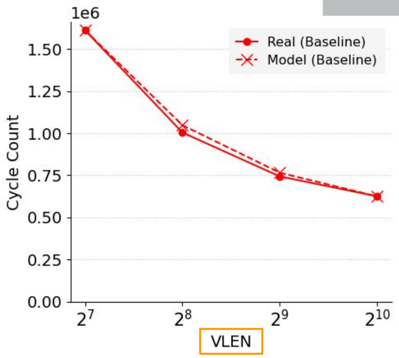
Datapath Width (DLEN) Model:

$$\varsigma' = \varsigma \cdot (1 - \rho \cdot f + \rho \cdot \frac{f \cdot \delta}{\delta'})$$

Vector Register Length (VLEN) Model:

$$\varsigma' = \varsigma \cdot (1 - \rho + \rho \cdot \frac{\nu}{\nu'})$$





, where Cycle (ς), scaled cycle (ς'), VLEN (ν), scaled VLEN (ν'), DLEN (δ), scaled DLEN (δ'), fraction of the parallel compute instruction (f), fraction of parallel compute cycles contributing to the arithmetic (ϱ).

Performance Models

Datapath Width (DLEN) Model:

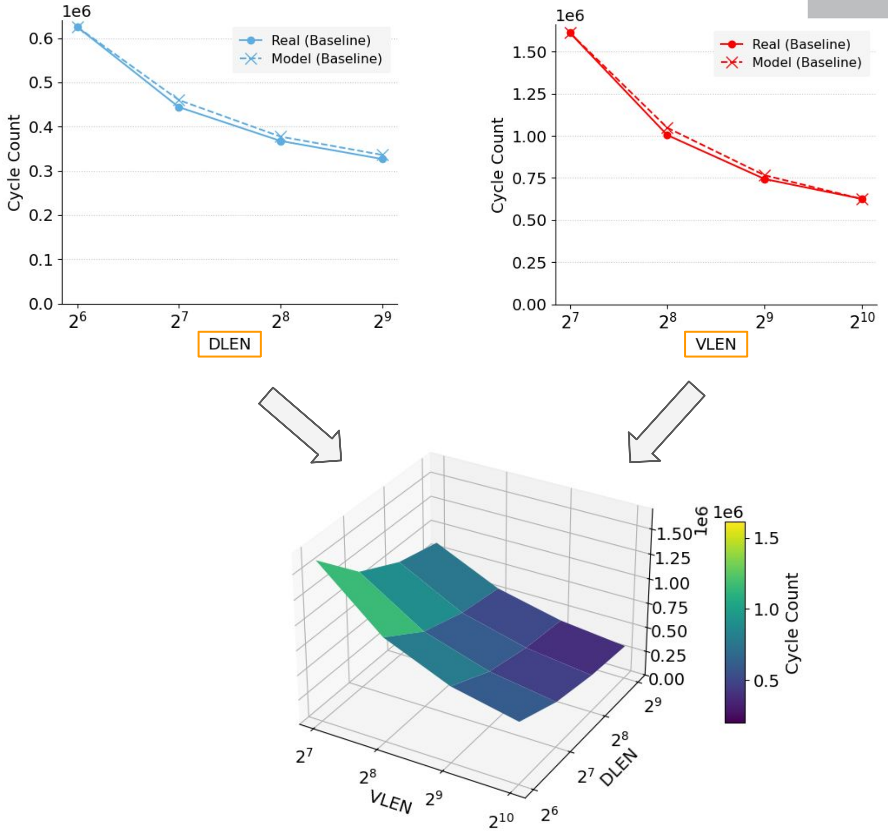
$$\varsigma' = \varsigma \cdot (1 - \rho \cdot f + \rho \cdot \frac{f \cdot \delta}{\delta'})$$

Vector Register Length (VLEN) Model:

$$\varsigma' = \varsigma \cdot (1 - \rho + \rho \cdot \frac{\nu}{\nu'})$$

3D Model Combining DLEN and VLEN:

$$\mathcal{M}_{3D} = \mathcal{M}_{DLEN} \cdot \mathcal{M}_{VLEN}$$



, where Cycle (ς), scaled cycle (ς'), VLEN (ν), scaled VLEN (ν'), DLEN (δ), scaled DLEN (δ'), fraction of the parallel compute instruction (f), fraction of parallel compute cycles contributing to the arithmetic (ϱ).

PISA: Performance Projection with Proxy ISA

Mapping each extended instruction to the most structurally similar RVV instruction.

Multi-word Extension
`udmvle64.v vd, (rs1), vm`
`vadcqc.vv vd, vmo, vs1, vs2, vmi`
`vmulf.vv vh, vl, vs1, vs2`
`vstack.vv op[1-3], vsp1, vsp2`

Zhang, N., Fu, S., & Franchetti, F. (2025). Towards Closing the Performance Gap for Cryptographic Kernels Between CPUs and Specialized Hardware. In Proceedings of MICRO 2025 (IEEE/ACM).

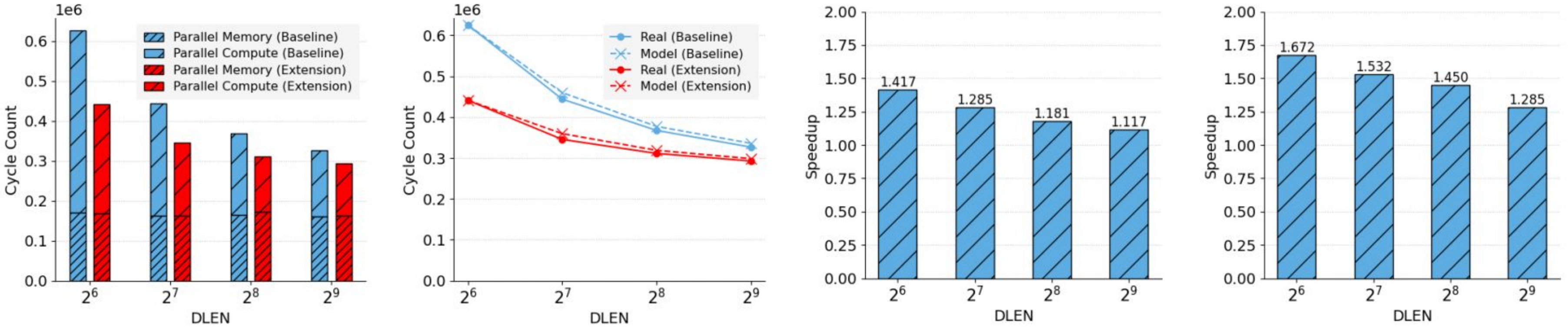
PISA: Performance Projection with Proxy ISA

Mapping each extended instruction to the most structurally similar RVV instruction.

Multi-word Extension	RVV proxy instruction
`udmvle64.v vd, (rs1), vm`	`vle64.v vd, (rs1), v0`
`vadcqc.vv vd, vmo, vs1, vs2, vmi`	`vadd.vv vs1, vs2, v0`
`vmulf.vv vh, vl, vs1, vs2`	`vmul.vv vs1, vs2, v0`
`vstack.vv op[1-3], vsp1, vsp2`	`vxor.vv vs1, vs2, v0`

Zhang, N., Fu, S., & Franchetti, F. (2025). Towards Closing the Performance Gap for Cryptographic Kernels Between CPUs and Specialized Hardware. In Proceedings of MICRO 2025 (IEEE/ACM).

Performance Results: Scaling DLEN



(a) Cycle Count Breakdown: Parallel Memory and Compute Components across Saturn Vector Unit Configurations with Varying DLEN.

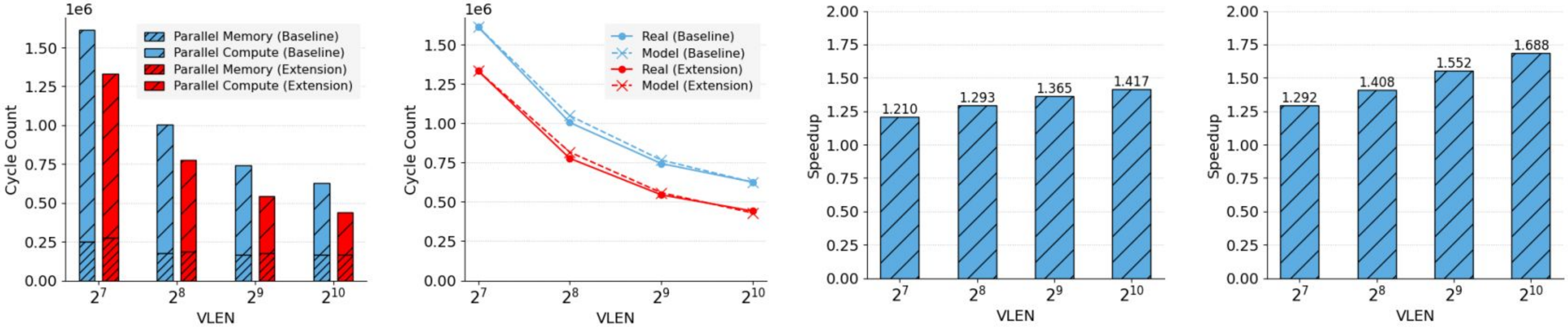
(b) Math Model for Total Cycle Count across Saturn Vector Unit Configurations with Varying DLEN.

(c) Overall Multi-word Extension Speedup Compared to Baseline Configurations with Varied DLEN.

(d) Multi-word Extension Speedup Compared to Baseline Configurations with Varied DLEN on Parallel Compute Part.

Figure 5: Performance analysis of baseline and extension when scaling DLEN.

Performance Results: Scaling VLEN



(a) Cycle Count Breakdown: Parallel Memory and Compute Components across Saturn Vector Unit Configurations with Varying VLEN.

(b) Math Model for Total Cycle Count across Saturn Vector Unit Configurations with Varying VLEN.

(c) Overall Multi-word Extension Speedup Compared to Baseline Configurations with Varied VLEN.

(d) Multi-word Extension Speedup Compared to Baseline Configurations with Varied VLEN on Parallel Compute Part.

Figure 6: Performance analysis of baseline and extension when scaling VLEN.

Takeaways

- Even with RVV's existing support for a global mask register to propagate carry masks, additional architectural extensions can further saturate the pipeline for HPC benchmarks.
- Performance gain is limited by different bottlenecks across different architectural configurations, making it worth studying how to maintain it.

Takeaways

- Even with RVV's existing support for a global mask register to propagate carry masks, additional architectural extensions can further saturate the pipeline for HPC benchmarks.
- Performance gain is limited by different bottlenecks across different architectural configurations, making it worth studying how to maintain it.

Q&A

Yunhao Lan:
yunhaolan@cmu.edu