# Enhancing convolutional neural network computation with integrated matrix extension

Chun-Nan. Ke, Heng-Kuan Lee, Yi-Xuan Huang

**Andes Technology**

RISC-V® Summit

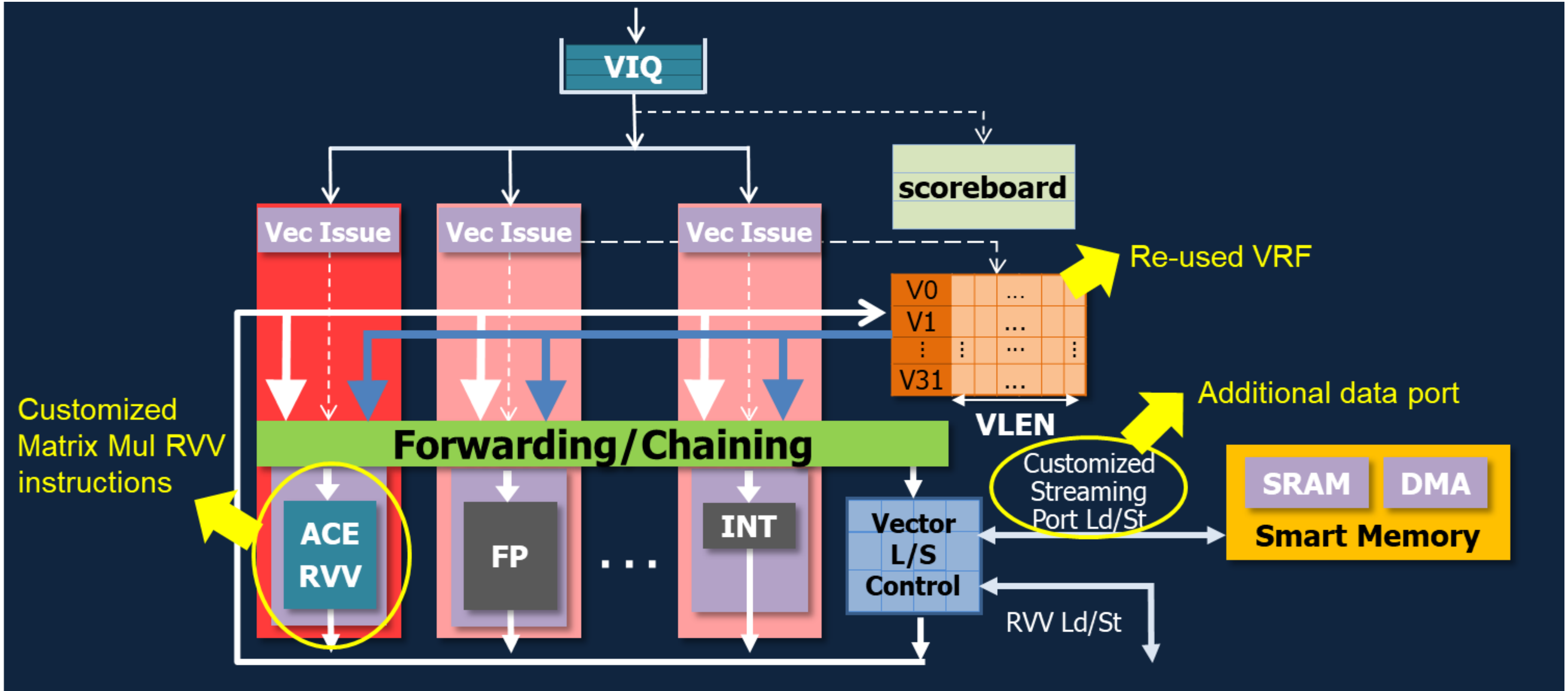
# Outline

- Motivation for Integrated-Matrix-Extension
- Matrix Operation Instruction Set Architecture
- Memory Sub-Sys Solution
- Zero Overhead Boundary Handling
- Performance Profiling/Sharing

# Motivation for Matrix Extension

- Explosive Computation Demand from AI/ML Has Driven the Innovation for Matrix/Tensor Processing Acceleration Techniques.
- A Novel Matrix Extension Aims for
  - Programmer-Friendly ISA design
  - Seamless data exchange with legacy RISC-V V-ext (RVV)
  - SW scalability with VLEN agnosticism.
  - Enhancing data locality with minimizing memory access power
  - Achieve optimal performance by maximizing MAC utilization-rate
  - Extreme low overhead for Matrix boundary handling

# Andes Custom Extension (ACE) on RVV



Re-used VRF

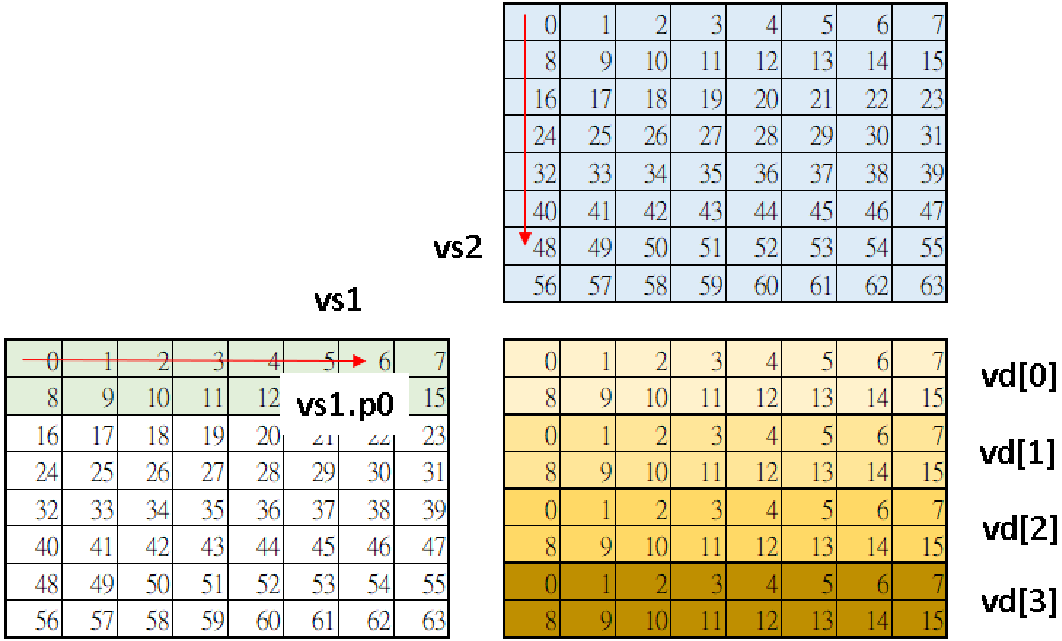
Additional data port

Customized Matrix Mul RVV instructions

**RISC-V custom extension unlocks the intensive computation of Matrix Multiplication in AI applications**

# Proposed Scalable Matrix-Mul-Acc Instruction

- Achieve SW Portability by Incorporating Following Innovations
  - Flexible management for vector registers
  - High Compute Intensity Performance for square outer products
  - Efficient to Corporate with Cache Memory Sub-sys
- amm vd, vb, va
  - vd[0] = vs1.p0 * vs2



vs2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |

vs1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | vs1.p0 | | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |

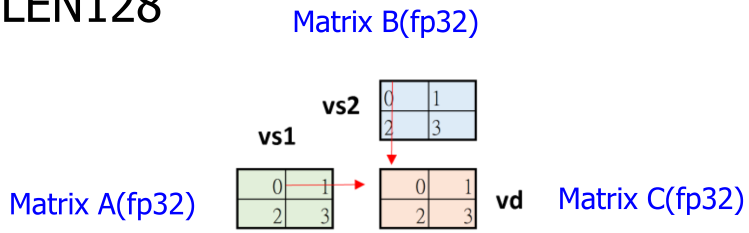
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | vd[0] |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | vd[1] |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | vd[2] |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | vd[3] |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |

Taking

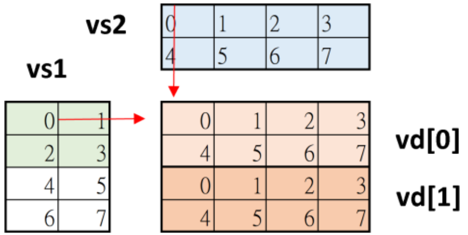
# Scalability Cross All VLENs (Widening 1x)

- Considering Floating Point Model as fp32 += fp32*fp32

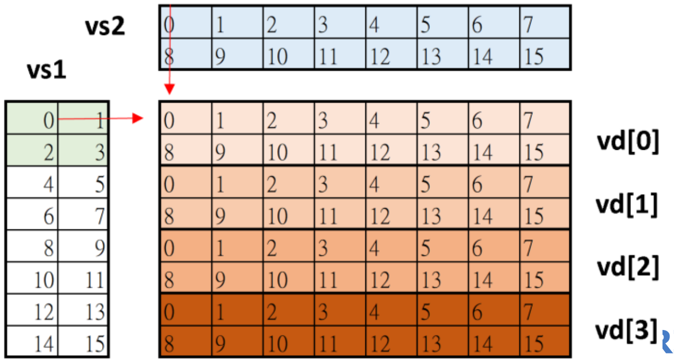
VLEN128



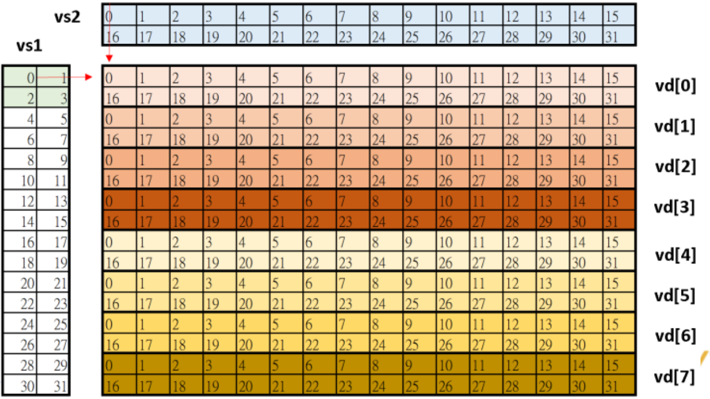
VLEN256



VLEN512



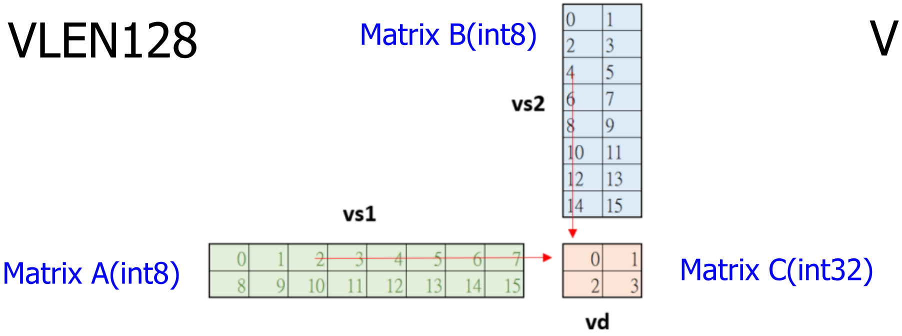
VLEN1024



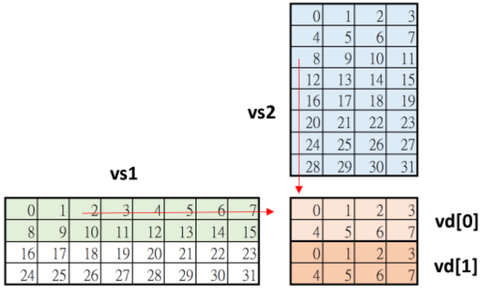
# Scalability Cross All VLENs (Widening 4x)

- Considering Quantized Model as int32 += int8*int8

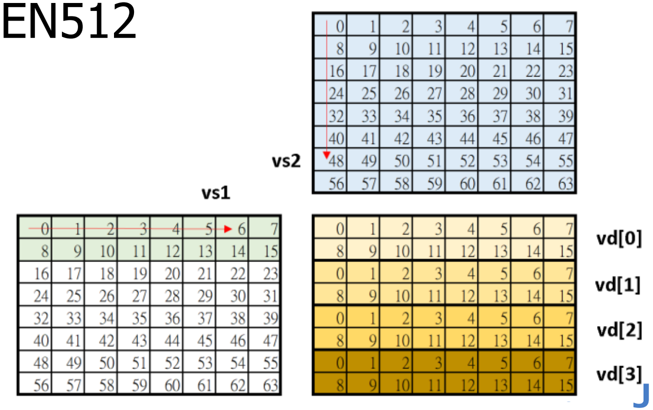
## VLEN128



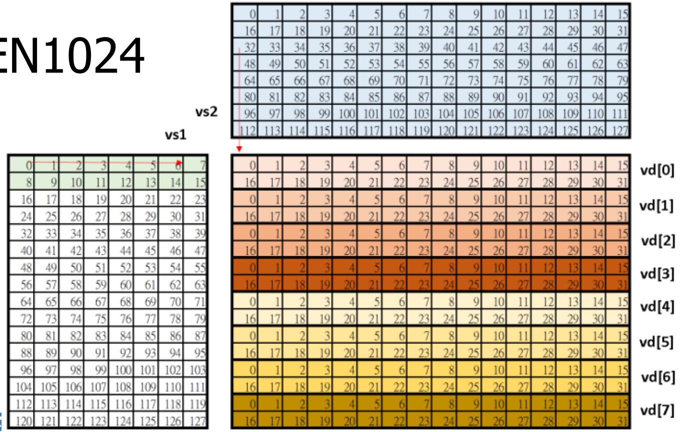
## VLEN256



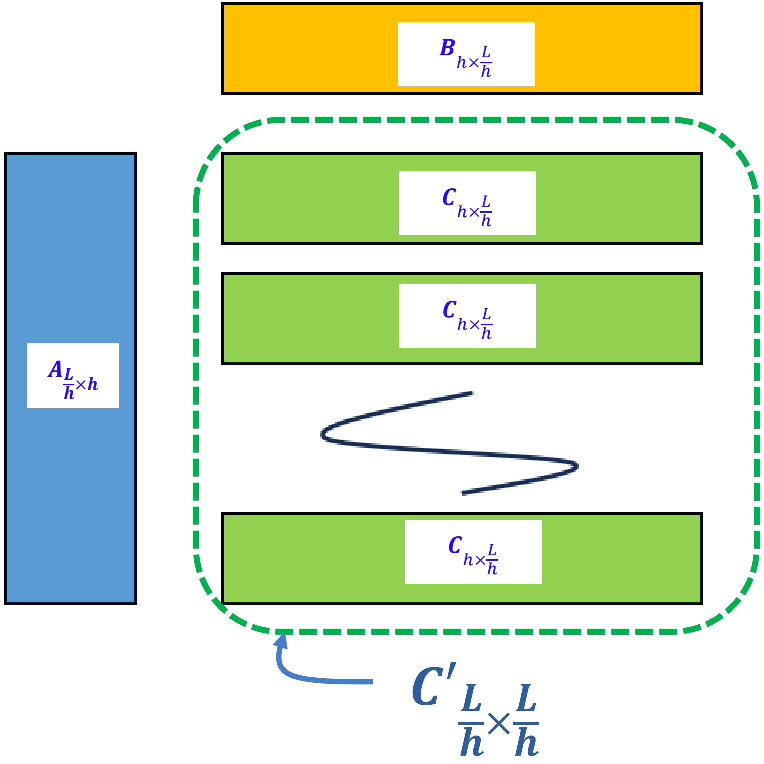
## VLEN512



## VLEN1024



# Architecture Gain for Compute-Intensity(1/)



① $C_{h\times\frac{L}{h}}$ Single VRF choose h for achieving optimal Architecture State Usages , where h=$\sqrt{L_{min}}$ keeping C efficiently utilized

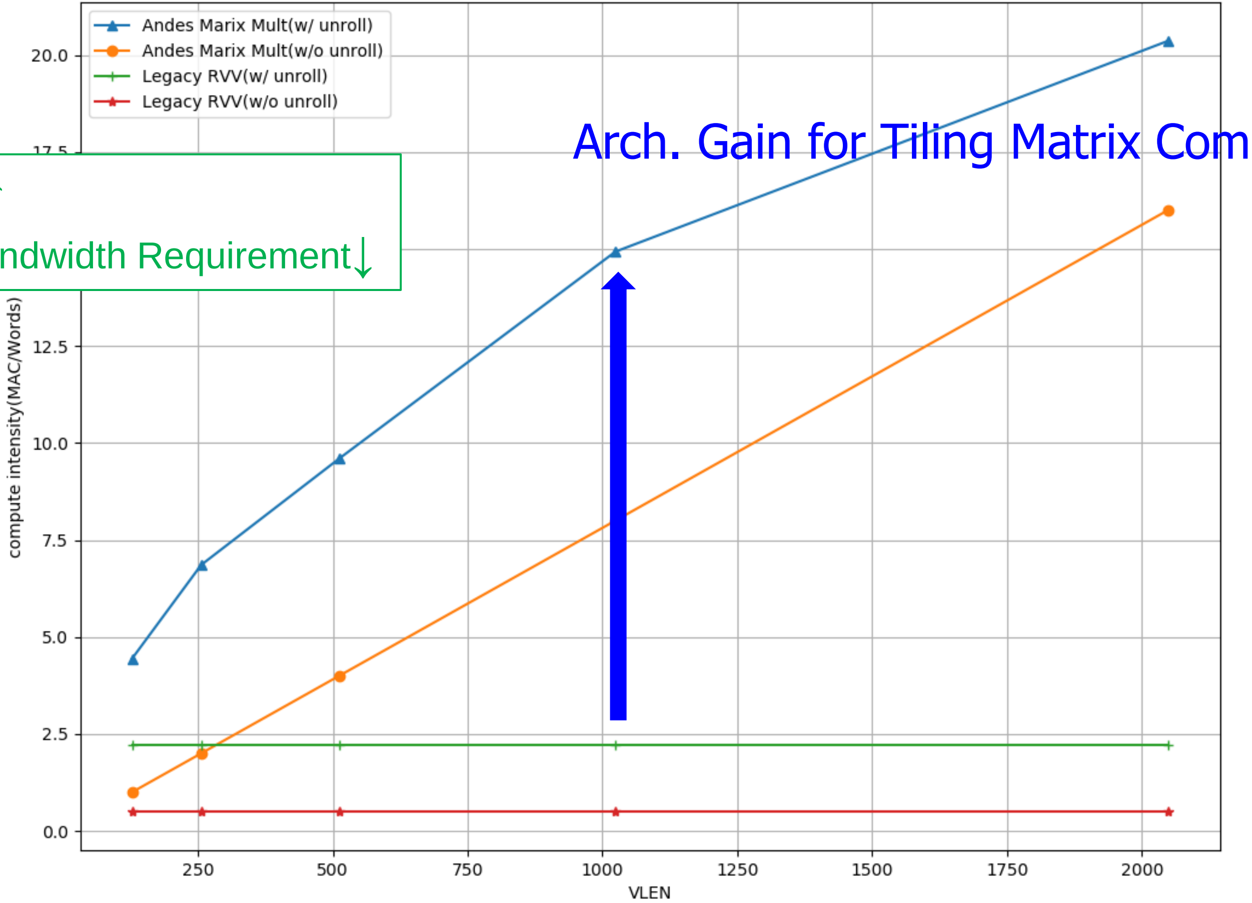
② $C'_{\frac{L}{h}\times\frac{L}{h}}$ ACC VRFs keep m = n as ACC square for achieving Optimal Computation Intensity

③ Near Optimal Compute Rate :$\tilde{M}\tilde{N}\left(\frac{L^2}{4*L_{min}}\right)$†

④ Near Optimal Compute Intensity:$\frac{\tilde{M}\tilde{N}\left(\frac{L}{\sqrt{L_{min}}}\right)^2*\sqrt{L_{min}}}{(\tilde{M}+\tilde{N})L}=\frac{\tilde{M}\tilde{N}L}{\sqrt{L_{min}}(\tilde{M}+\tilde{N})}$†

†: where $\tilde{M},\tilde{N}$ are reasonable implementation factor (see Appendix for details)

# Architecture Gain for Compute-Intensity(2/)



Compute-Intensity↑

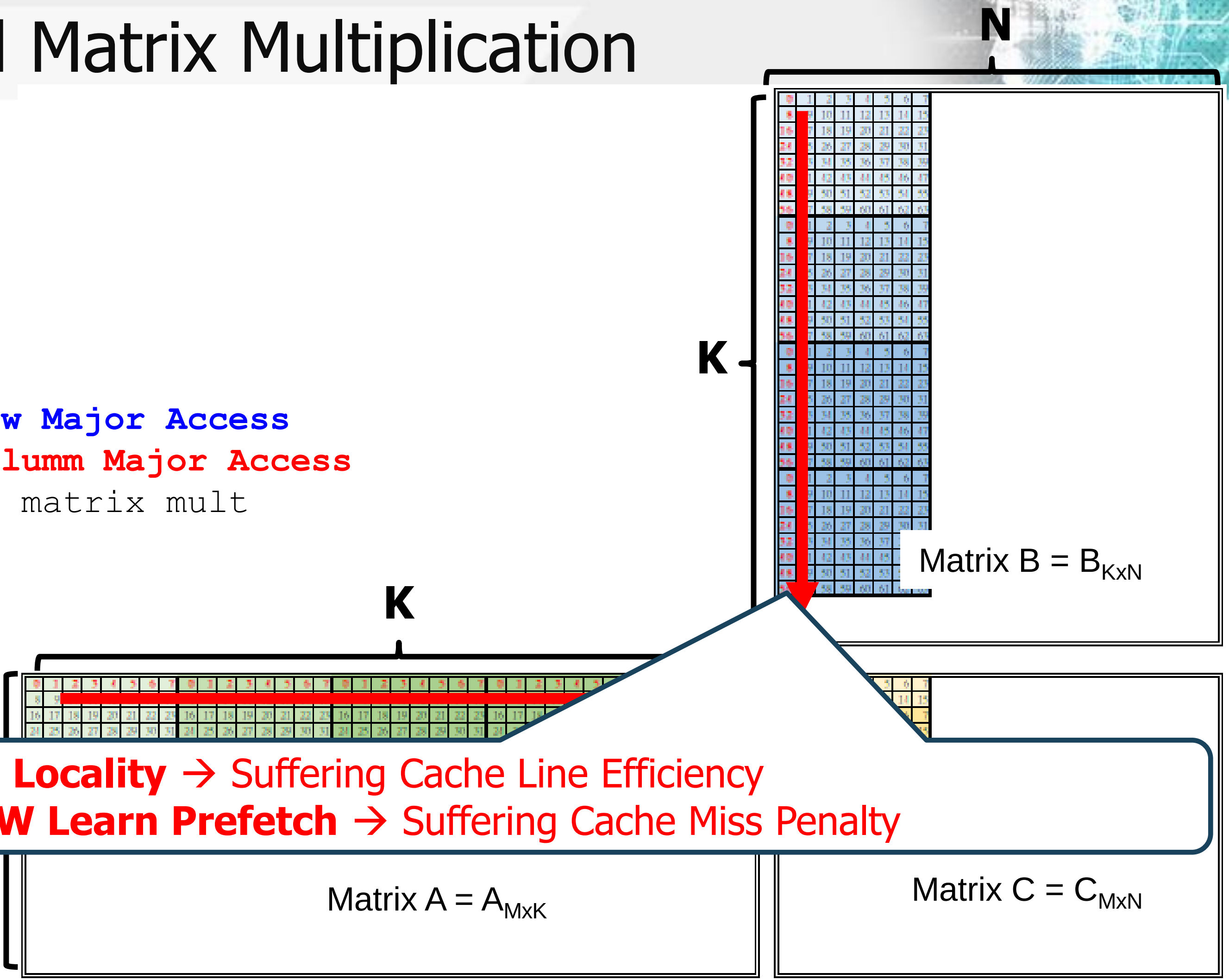
Memory Access Bandwidth Requirement↓

Arch. Gain for Tiling Matrix Computation

# Naive Tile-based Matrix Multiplication

$C_{MxN}+=A_{MxK}*B_{KxN}$

```
void kernel() {
…
 while(k>0){
  vld.2d va,[mem_a];//Row Major Access
  vld.2d vb,[mem_b];//Columm Major Access
  amm vc, vb, va; //Tile matrix mult
  k-=Ktile;
 }
…
}
```

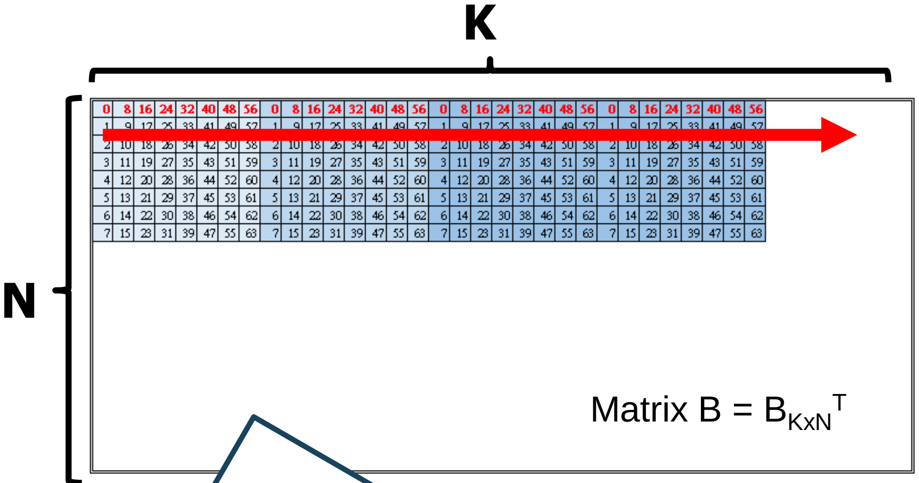


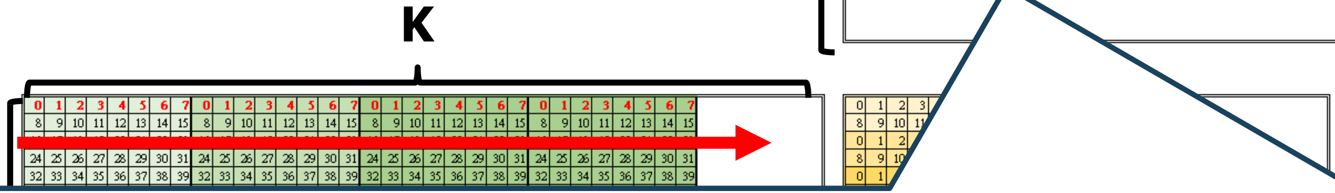
(1) Non-friendly for **Cache Locality** → Suffering Cache Line Efficiency
(2) Non-Easy for **Cache HW Learn Prefetch** → Suffering Cache Miss Penalty

# Efficient Tile-based Matrix Multiplication

$C_{MxN} += A_{MxK} * B_{KxN} += A_{MxK} * B_{NxK}^{T}$

```
void kernel() {
…
 while(k>0){
  vld.2d vb,[mem_b];//Row Major Access
  vt vb vb;//in-place VRF transpose
  vld.2d va,[mem_a];//Row Major Access
  amm vc, vb, va; //Tile matrix mult
  k-=Ktile;
 }
…
}
```



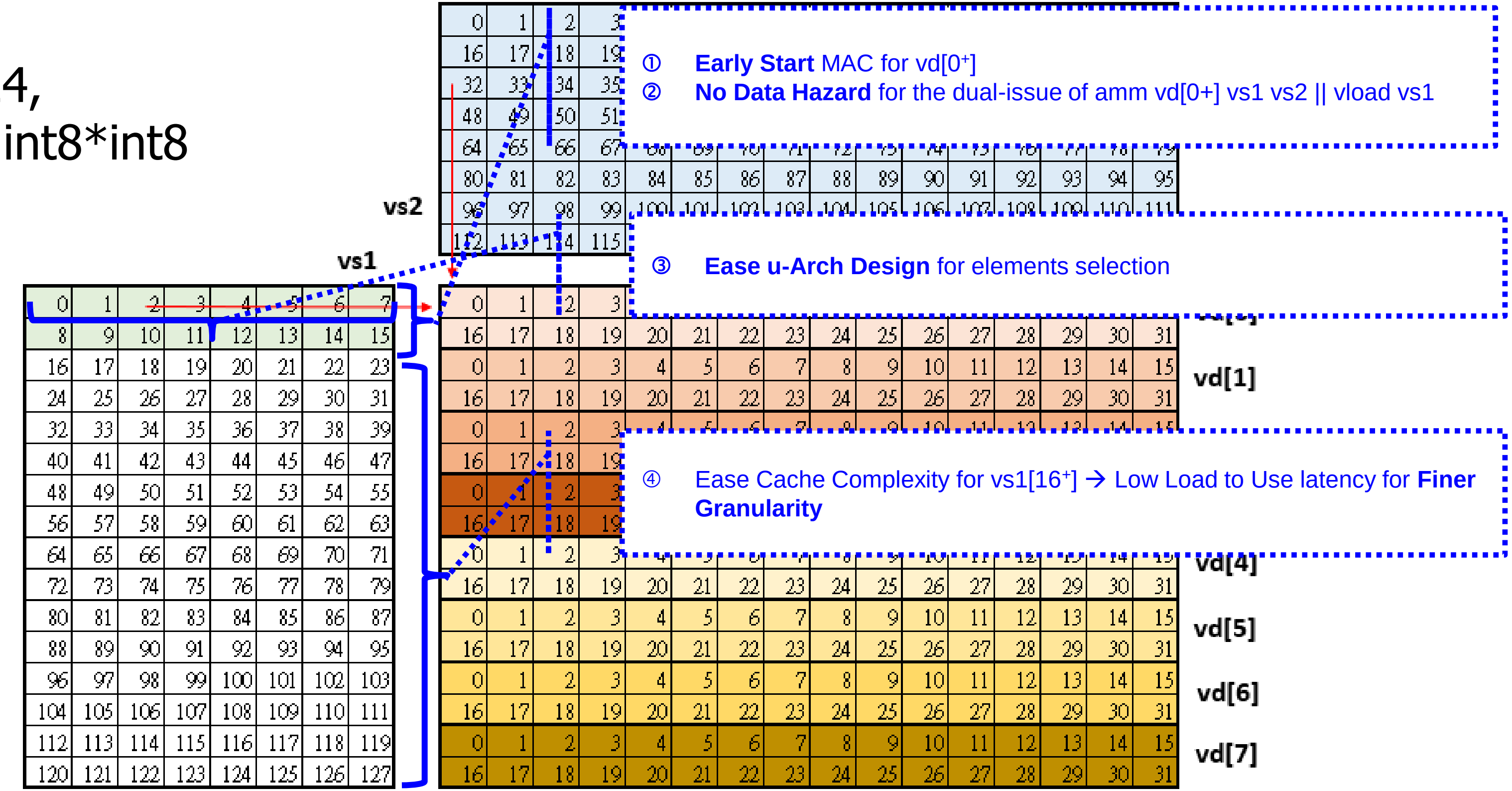


(1) Friendly for **Cache Locality** → Gain Cache Line Efficiency
(2) Easy for **Cache HW Learn Prefetch** → Reduce Cache Miss Penalty
(3) VRF transpose is easy to dual-issue with non-dependent instruction → No Overhead

Matrix A = $A_{MxK}$

# Effective (Emul Aware) 2D-load

VLEN 1024,
int32 += int8*int8



① **Early Start** MAC for vd[0⁺]

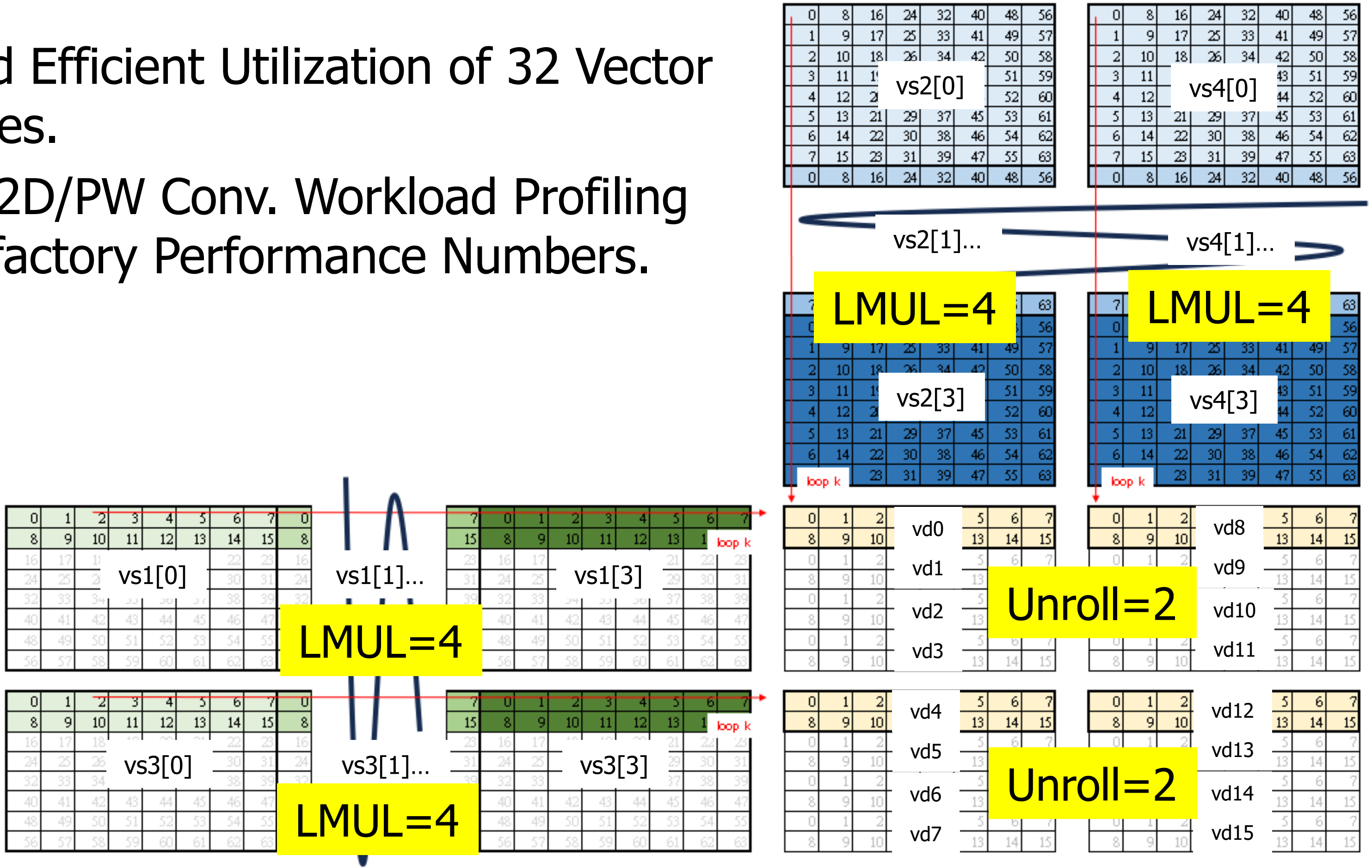
② **No Data Hazard** for the dual-issue of amm vd[0+] vs1 vs2 || vload vs1

③ **Ease u-Arch Design** for elements selection

④ Ease Cache Complexity for vs1[16⁺] → Low Load to Use latency for **Finer Granularity**

# LMUL/Unroll Support for 2D-LSU

- Elegant and Efficient Utilization of 32 Vector Register Files.
- GeMM/Con2D/PW Conv. Workload Profiling show Satisfactory Performance Numbers.



# 2D-LSU Programming Model for Opt. Cache Sub-sys

```
MGRP = VLEN >> exp(min_segs);
void gemm_kernel() {
…
 vector<int8> va,vb;
 vector<int32> vd[MGRP];
 vsetvl_lmul(4)
 vld.2d vs2,rs2,rs4,imms;
 vld.2d vs4,rs6,rs8,imms;
 while(k>0){
   vld.ef vs1,rs1,rs3,imms;
   amm   vd0,vs2[0],vs1[0];
   amm   vd0,vs2[1],vs1[1];
   amm   vd0,vs2[2],vs1[2];
   amm   vd0,vs2[3],vs1[3]||vld.ef vs1,rs1,rs3,imms;
   amm   vd0,vs2[0],vs1[0];
   amm   vd0,vs2[1],vs1[1];
   amm   vd0,vs2[2],vs1[2];
   amm   vd0,vs2[3],vs1[3]||vld.ef vs1,rs1,rs3,imms;
 …unroll…
   vld.2d vs2,rs2,rs4,imms;
   vld.2d vs4,rs6,rs8,imms;
   k -= Ktile*unroll_factor;
 }
 vsetvl_lmul(8)
 vst.2d vd,rs9,rs10,imms;
}
```

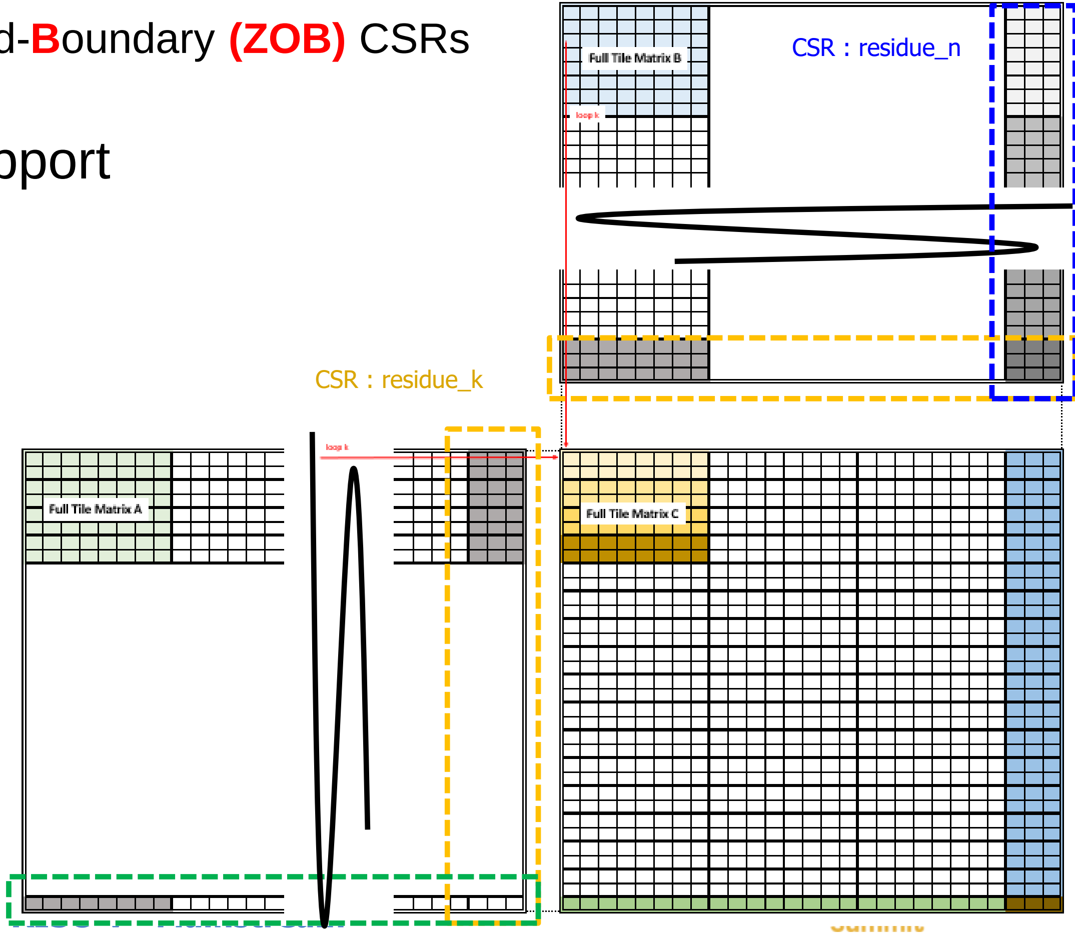
+ Matrix B (vs2/vs4) Transpose Support → Exhibits Cache Locality Performance
+ LMUL Support for 2d-load → Mitigating Memory Access Latency with Register Grouping LMUL Support

+ Novel Effective Load → Enhance Cache Latency with dual-issue

+ LMUL Support for 2d-store or Seamless data exchange with RVV

# Zero-Overhead-Boundary Support(1/)

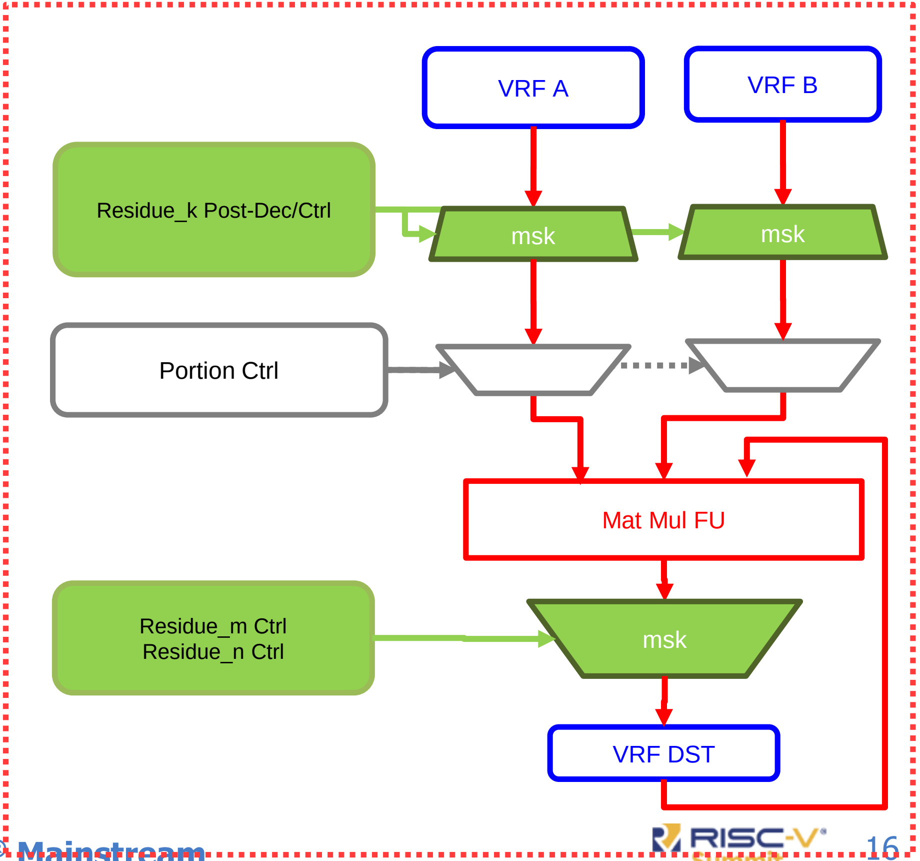
- Elegant and Easy Config **Z**ero-**O**verhead-**B**oundary **(ZOB)** CSRs Assisted
  - □Novel Multi-Dimensional Support
  - □Residue_n
  - □Residue_k
  - □Residue_m



# Zero-Overhead-Boundary Support (2/)

- New Novel ISA-Coordination-Aware Fractured Matrix Computation.
  - □ Simple/Concise Function Unit Architecture
  - □ Low Cost HW Auto Decrement
  - □ Flexible Programing Model

| Solution | Speedups |
|---|---|
| Naiive Boundary Method | 1.0x |
| Zero-Overhead Boundary Handling | > 1.09x† |



†: Speedup ratio depends on Matrix Sizes, where speedups {2.36x, 1.36x, 1.09x} when square matrix scales around {128x128, 512x512, 2048x2048}, respectively.

# GeMM/Con2D

- GeMM 128x128x128

| Architecture<br>(VLEN/DLEN/AMM 512/512/512) | Speed-up | U-rate(%) |
|---|---:|---:|
| Std. RVV (libvec) | 1x | ~15% |
| AMM (w/o unroll) | ~1.8x | ~39% |
| **AMM (optimal w/ unroll,lmul)** | **~3.6x**† | **~82%** |

- Con2D

| Scenarios | Speed-up‡ | U-rate(%) |
|---|---:|---:|
| Con2D_0 | ~**3.7x** | ~77% |
| Con2D_1 | ~**3.1x** | ~82% |
| Con2D_2 | ~**5.2x** | ~78% |

†:based on 1-core configuration

‡:Std. RVV and AMM both data are shuffled in VDLM (HVM, latency is config as 3T)

# Convolutional Neural Network

- Projected Performance[†] Based on Mobilenetv1 Model
  - [https://arxiv.org/pdf/1704.04861](https://arxiv.org/pdf/1704.04861)
  - MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications
- Matrix + EdgeTrim[‡] Extensions Delivers High Efficient MAC Utilization Rates, especially for
  - Distinct Pointwise Convolution Layers
  - Asymmetrical Post-Convolution Quantization Operators

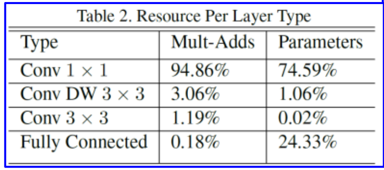


Table 2. Resource Per Layer Type

| Type | Mult-Adds | Parameters |
|---|---|---|
| Conv 1 × 1 | 94.86% | 74.59% |
| Conv DW 3 × 3 | 3.06% | 1.06% |
| Conv 3 × 3 | 1.19% | 0.02% |
| Fully Connected | 0.18% | 24.33% |

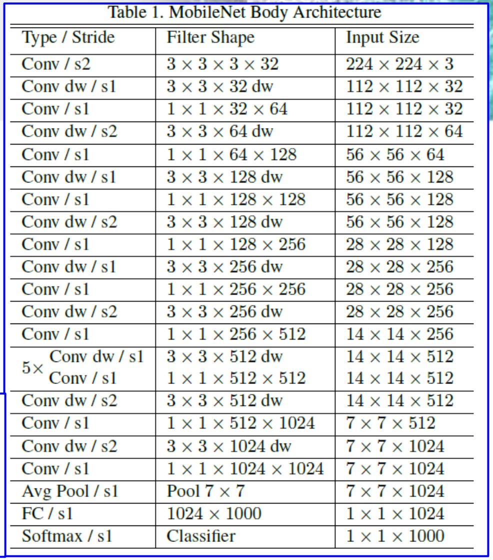
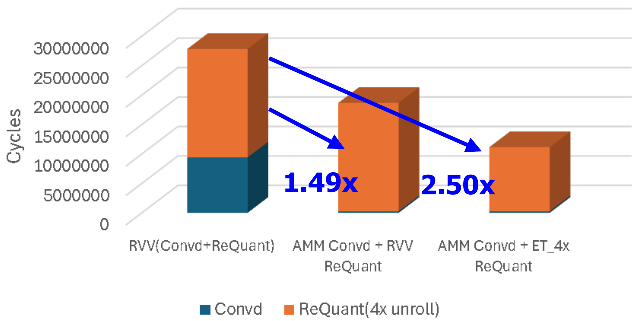


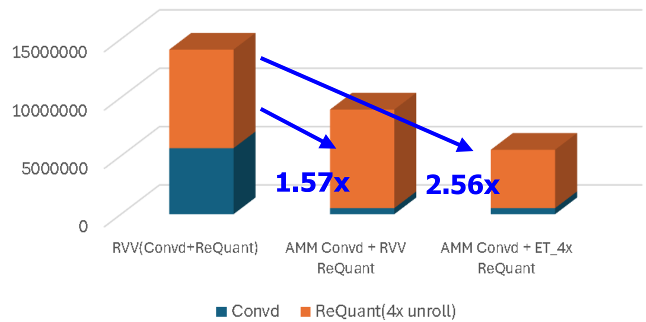
Table 1. MobileNet Body Architecture

| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | 3 × 3 × 3 × 32 | 224 × 224 × 3 |
| Conv dw / s1 | 3 × 3 × 32 dw | 112 × 112 × 32 |
| Conv / s1 | 1 × 1 × 32 × 64 | 112 × 112 × 32 |
| Conv dw / s2 | 3 × 3 × 64 dw | 112 × 112 × 64 |
| Conv / s1 | 1 × 1 × 64 × 128 | 56 × 56 × 64 |
| Conv dw / s1 | 3 × 3 × 128 dw | 56 × 56 × 128 |
| Conv / s1 | 1 × 1 × 128 × 128 | 56 × 56 × 128 |
| Conv dw / s2 | 3 × 3 × 128 dw | 56 × 56 × 128 |
| Conv / s1 | 1 × 1 × 128 × 256 | 28 × 28 × 128 |
| Conv dw / s1 | 3 × 3 × 256 dw | 28 × 28 × 256 |
| Conv / s1 | 1 × 1 × 256 × 256 | 28 × 28 × 256 |
| Conv dw / s2 | 3 × 3 × 256 dw | 28 × 28 × 256 |
| Conv / s1 | 1 × 1 × 256 × 512 | 14 × 14 × 256 |
| 5× Conv dw / s1 | 3 × 3 × 512 dw | 14 × 14 × 512 |
| Conv / s1 | 1 × 1 × 512 × 512 | 14 × 14 × 512 |
| Conv dw / s2 | 3 × 3 × 512 dw | 14 × 14 × 512 |
| Conv / s1 | 1 × 1 × 512 × 1024 | 7 × 7 × 512 |
| Conv dw / s2 | 3 × 3 × 1024 dw | 7 × 7 × 1024 |
| Conv / s1 | 1 × 1 × 1024 × 1024 | 7 × 7 × 1024 |
| Avg Pool / s1 | Pool 7 × 7 | 7 × 7 × 1024 |
| FC / s1 | 1024 × 1000 | 1 × 1 × 1024 |
| Softmax / s1 | Classifier | 1 × 1 × 1000 |

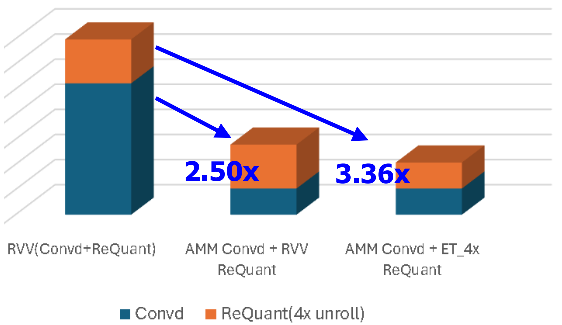
layer 3, in_ch=32
Convd:ReQuant ratio 1 : 1.95



layer 7, in_ch=128
Convd:ReQuant ratio 1 : 1.49



layer 27, in_ch=1024
Convd:ReQuant ratio 1 : 0.33



[†]:Estimated by Matrix-Multiplication U-rate (whole model on fpga measurement is underworking)

[‡]:EdgeTrim: Acceleration Instructions Specifically for ReQuantization Operators

# Conclusion

- A Novel Integrated Matrix Extension is proposed, Including:
  - Integrated-Facility for Seamless Data Exchange with RVV
  - SW Scalability with VLEN Agnosticism
  - Enhanced Computation Intensity to Minimize Memory IO Power
  - Achieve Optimal Performance with Maximizing MAC U-rate
  - 2D-Load Store Unit + Cache Sub-sys to Accommodate Data Access Throughput
  - Multi-dimensional Zero-Overhead Boundary Handling
- Profiling/Projected Results Demonstrate Significant Performance Improvement