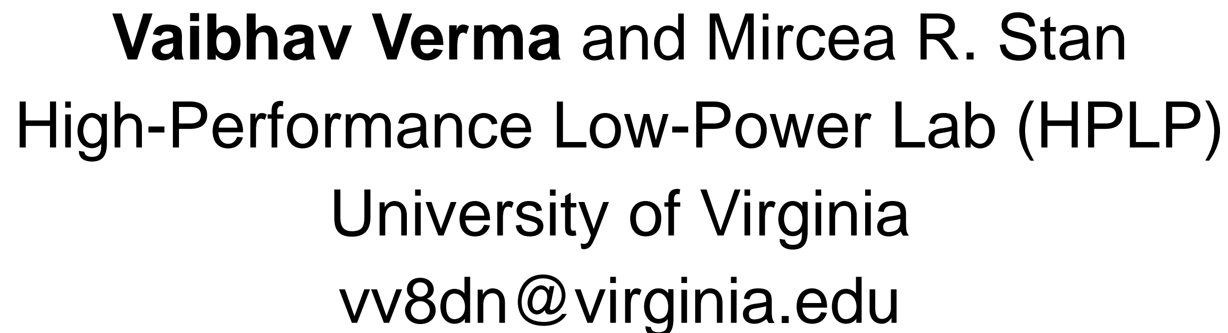
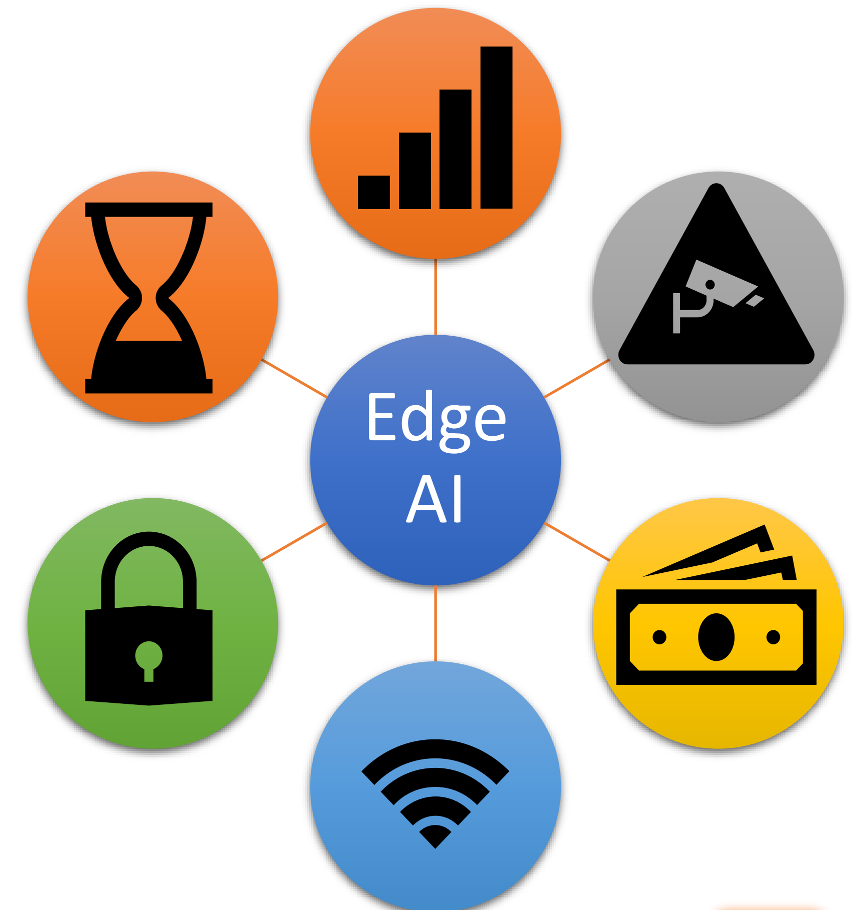
# **AI-RISC** - Scalable RISC-V Processor with Tightly Integrated AI Accelerators and Custom Instruction Extensions


**Vaibhav Verma** and Mircea R. Stan
High-Performance Low-Power Lab (HPLP)
University of Virginia
vv8dn@virginia.edu

# Need for Edge AI

- **Latency** – enable real-time AI systems
- **QoS** – cannot rely on connectivity in remote areas
- **Security** – sending data over network not secure
- **Privacy** – keep private data locally on device
- **Bandwidth** – send "information" to cloud rather than "data"
- **Cost** – data communication is costly



hplp High-Performance Low-Power
University of Virginia

# But Edge AI is different!

- Smaller neural network models
- Smaller batch sizes (≈ 1)
- Edge devices are cost, area and size limited
- Edge devices need to support both AI and non-AI applications
- Edge processors lack support for Keras, PyTorch, MXNet etc.

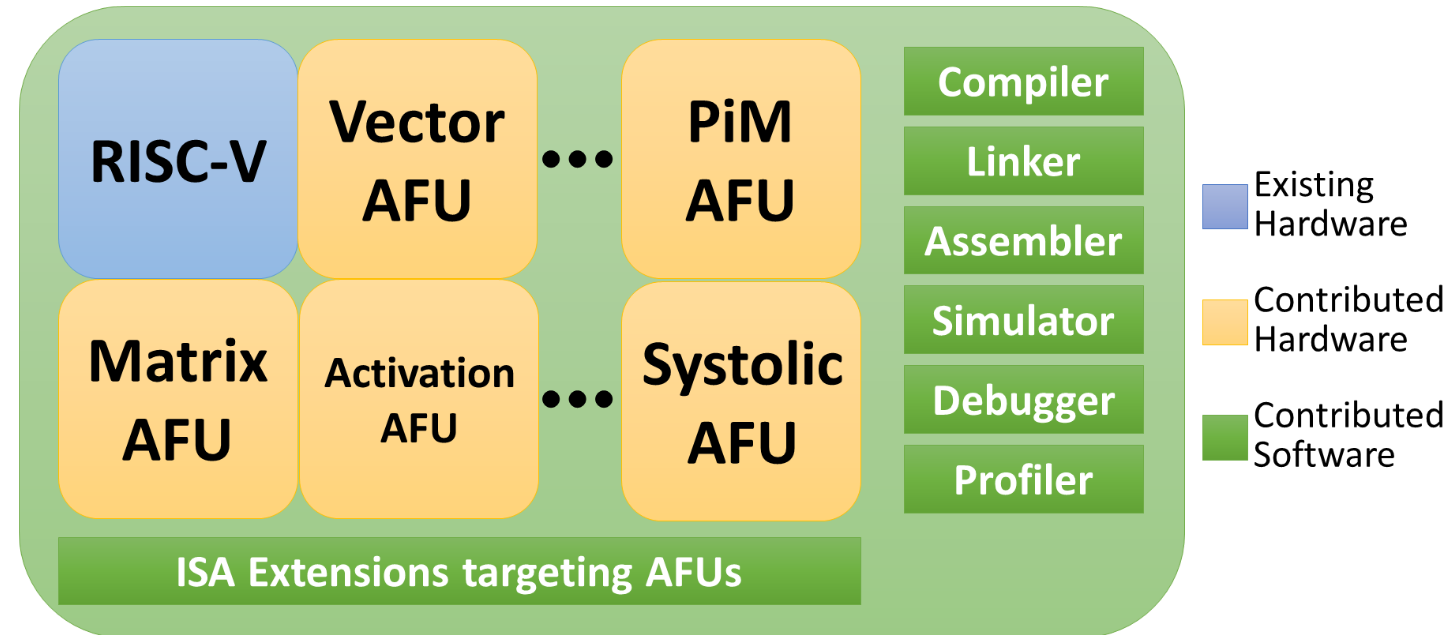
# AI-RISC

Custom RISC-V processor with ISA extensions targeting AI applications

Tightly integrated AI accelerators for fine-grained offloading of AI tasks

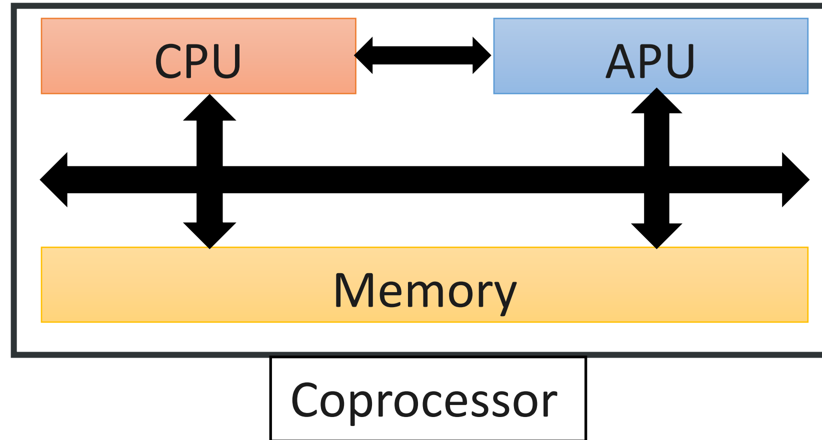
End-to-end hardware/software co-design solution

Support for AI and non-AI applications on the same processor

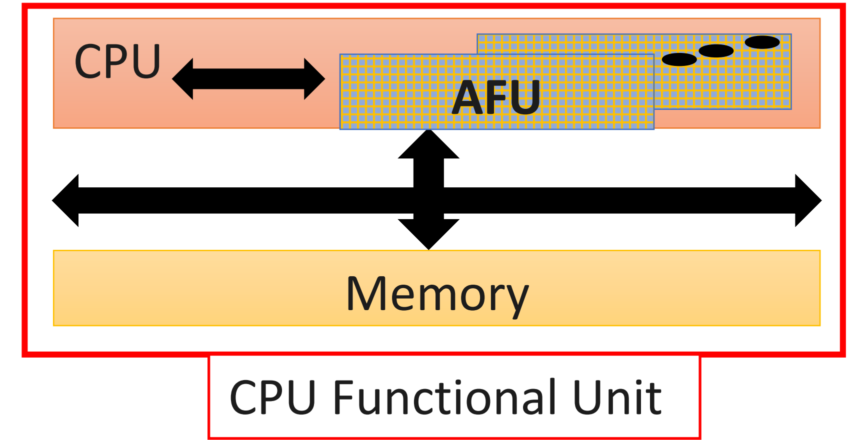


# AI Functional Unit (AFU)

**APU** = Coarse-grained **AI** **P**rocessing **U**nit

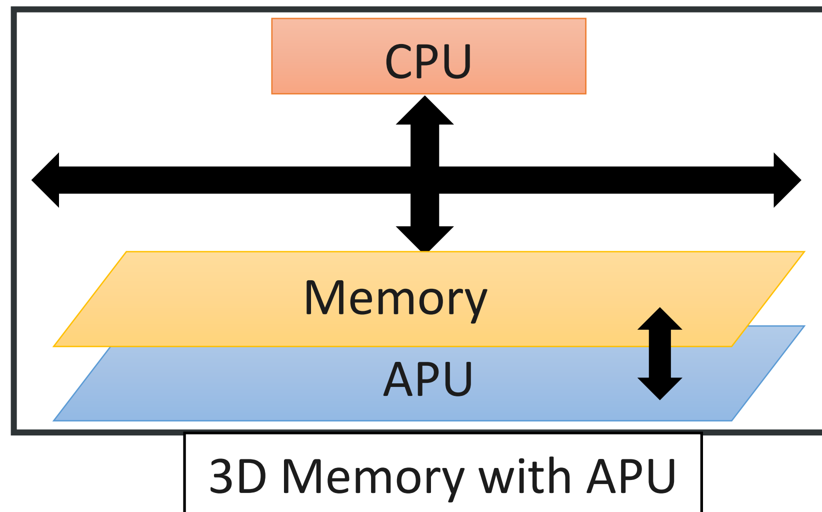



Coprocessor

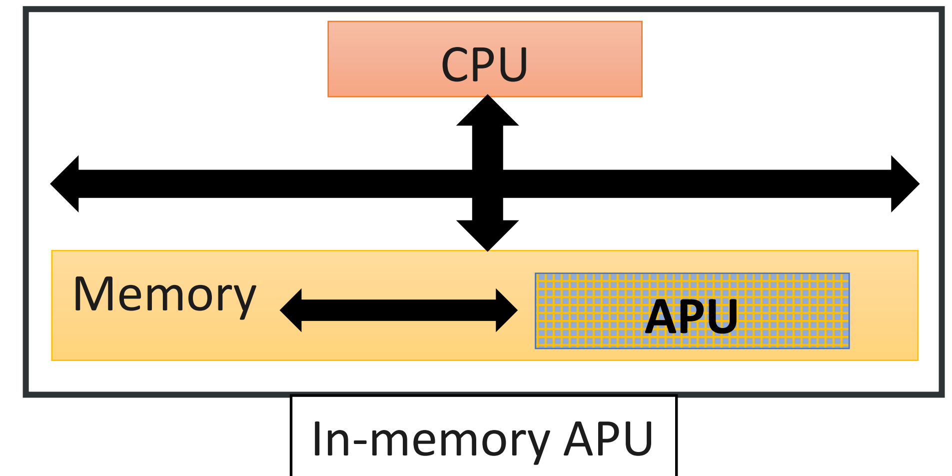



CPU Functional Unit

**AFU** = Fine-grained **AI** **F**unctional **U**nit



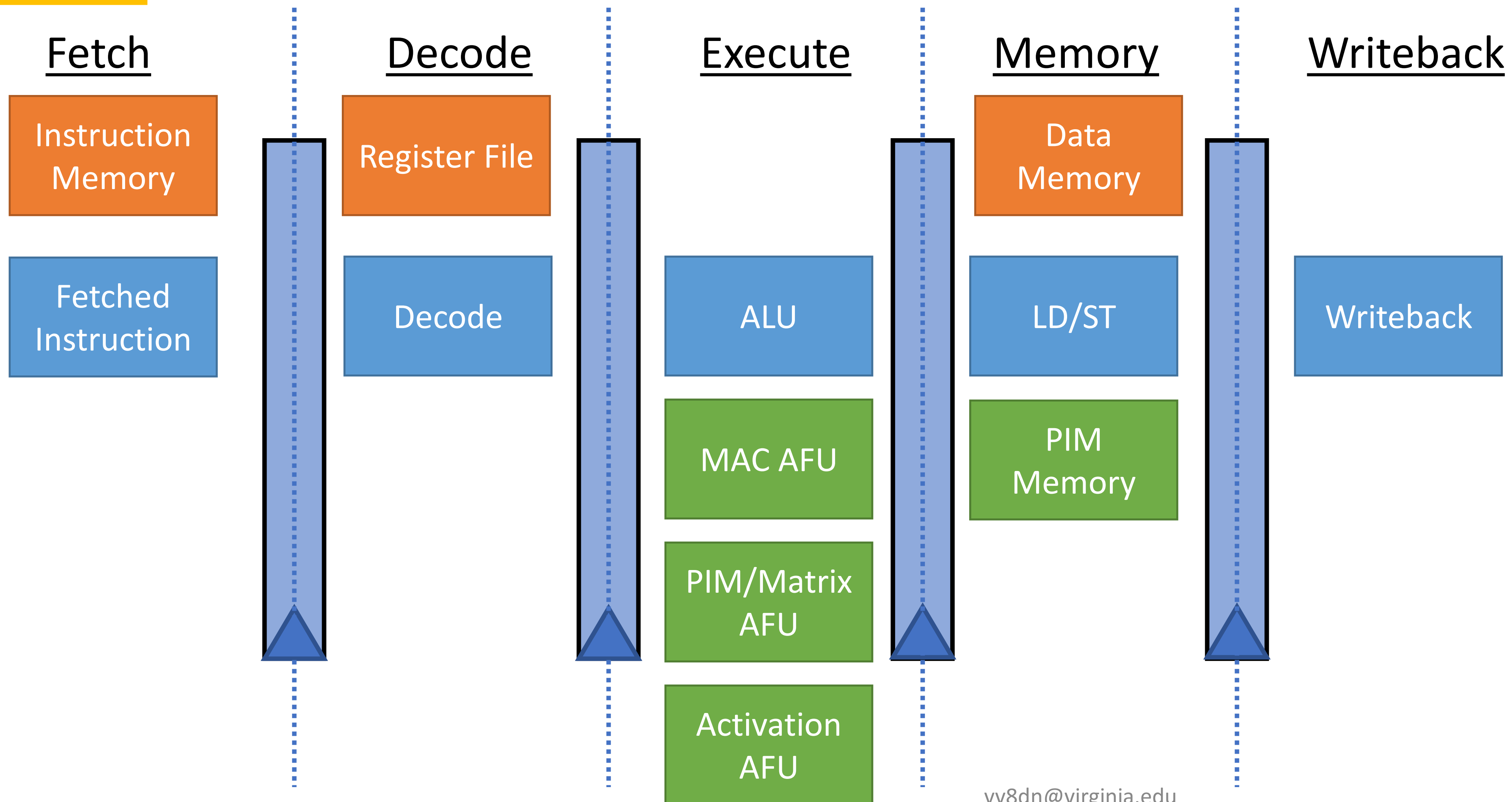

3D Memory with APU




In-memory APU

Image adapted from PRIME, ISCA 2016

# Tightly Integrated AFU




vv8dn@virginia.edu

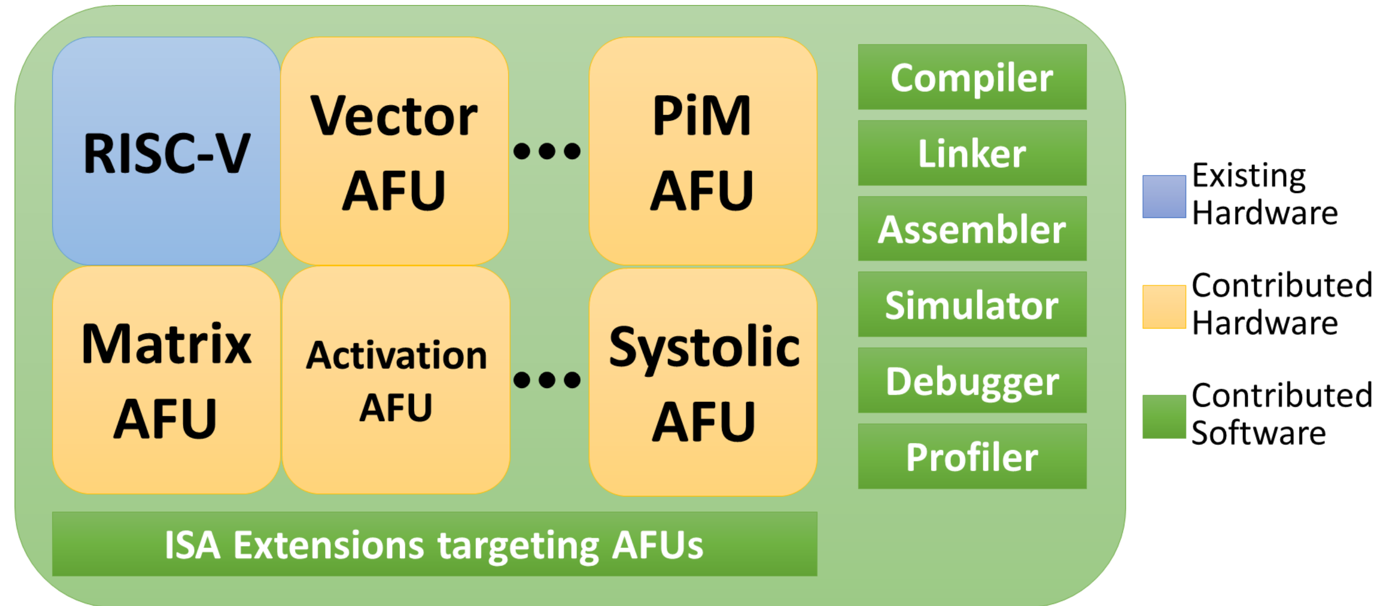
# AI extensions

Grouped by corresponding AI Functional Units (AFU)

- Instructions in **bold** are already implemented
- All instructions utilize RISC-V CUSTOM-2 opcode space

# AI extensions – General

- **MAC**
- **Packed SIMD MAC**
- Quantize
  - Accelerates cast and recast
  - Accelerate conditional clipping
- Transpose
  - Accelerates data layout changes
- **Post increment ld/st**
  - Already implemented in ASIP Designer RISC-V example
- **Hardware loops**
  - Already implemented in ASIP Designer RISC-V example

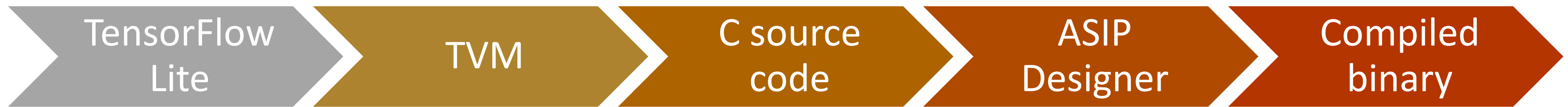


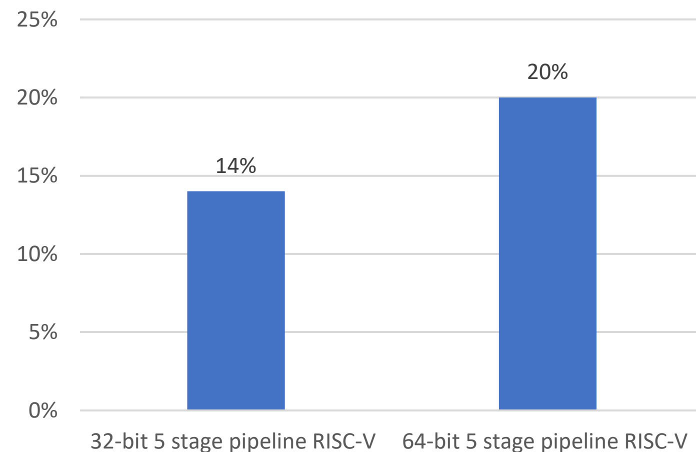
# Preliminary Results with MAC Instruction

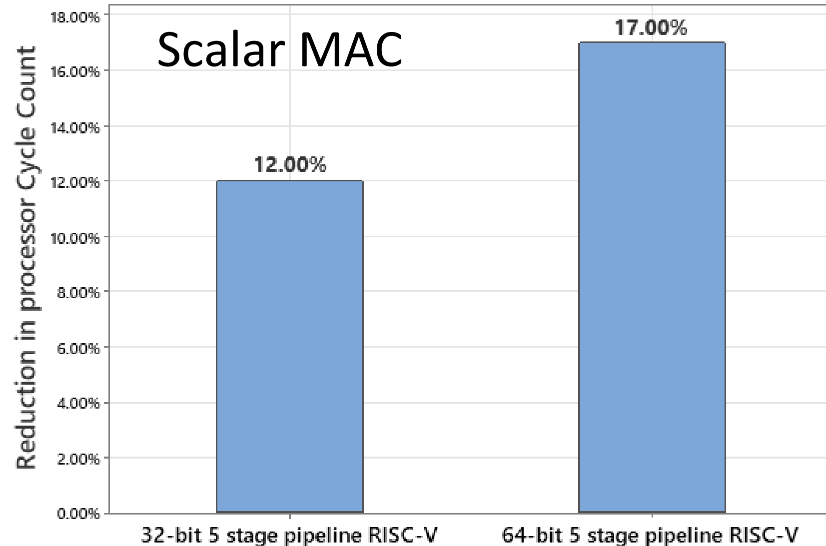
hplp High-Performance Low-Power | University of Virginia

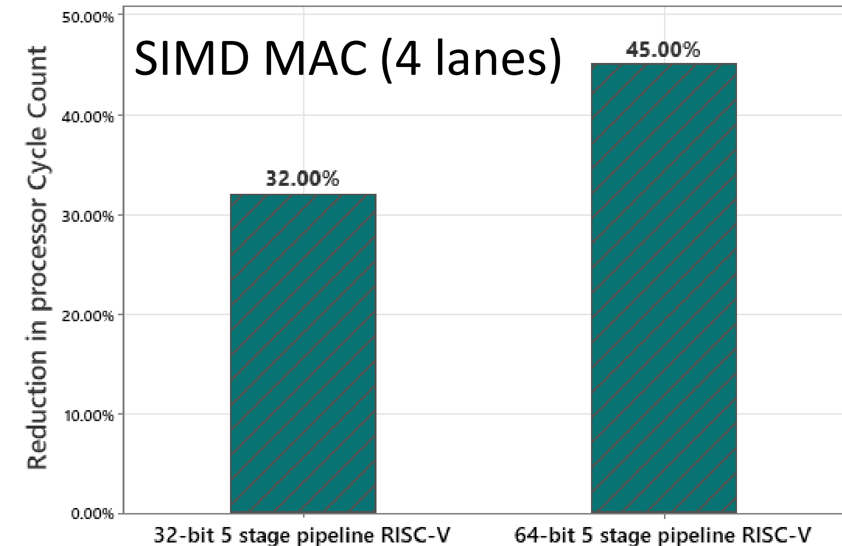
ResNet-8 TinyMLPerf Benchmark network on CIFAR-10 dataset

Quantized to int8



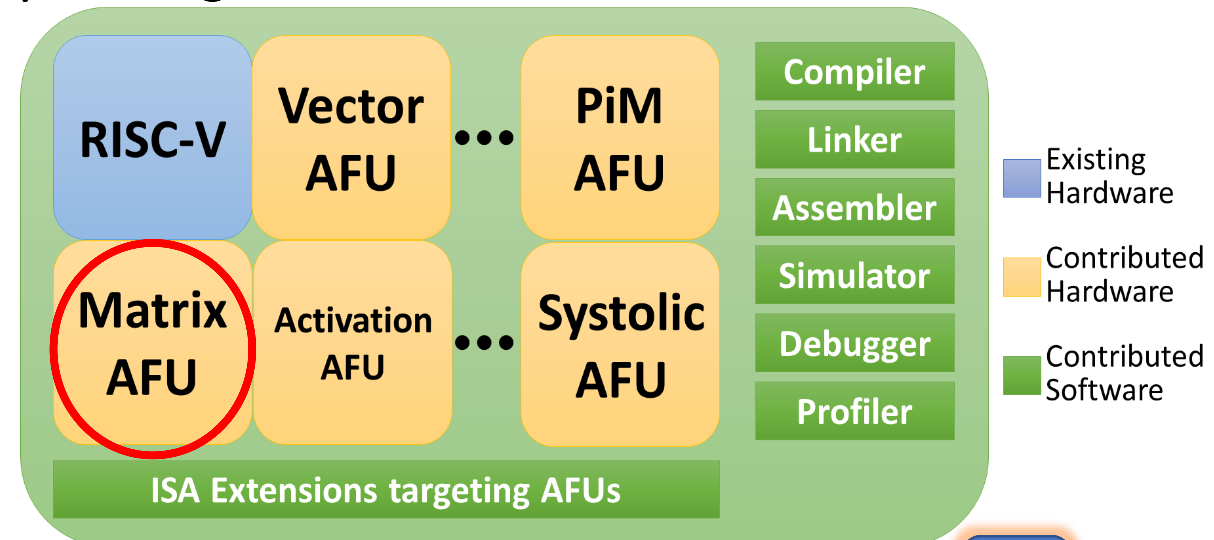



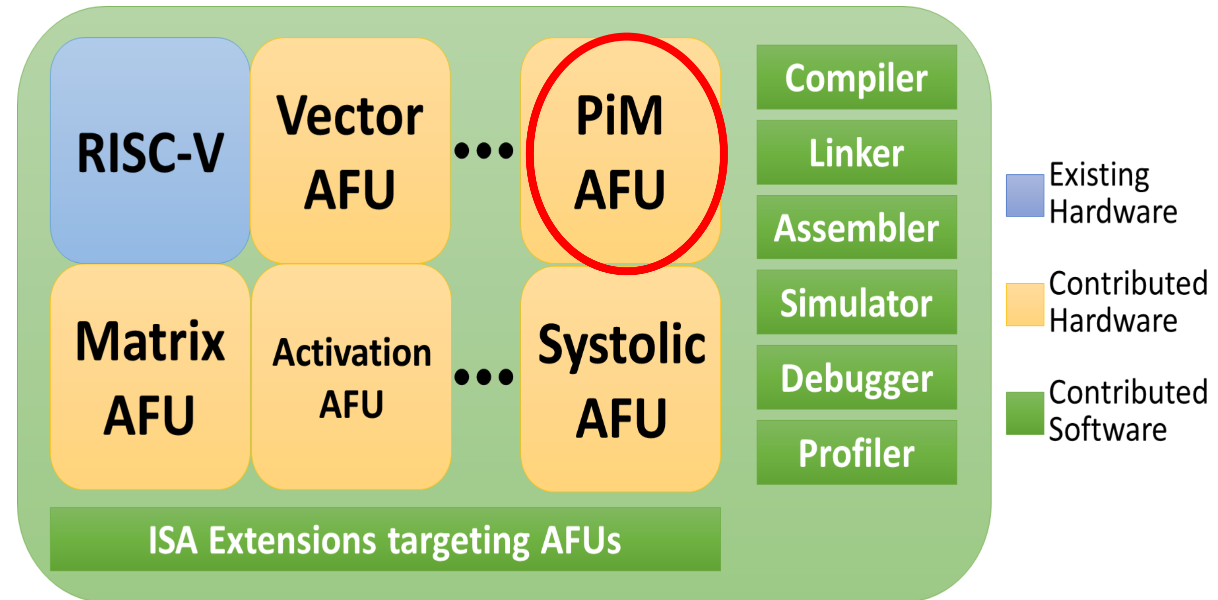
# AI extensions – Matrix AFU

- **Matrix data type**
  - Vector of 8 elements with each element of 8 bits → 64 bits wide
- **GEMM instructions**
  - 2x4 * 4x2 → 2x2 result, 16 bits
  - Planned – 4x2 * 2x4 – require double/quad register for result
- **GEMV instructions**
  - 1x4 * 4x2
  - 2x4 * 4x1
- **Vector-Vector multiplication**
  - 1x8 * 8x1

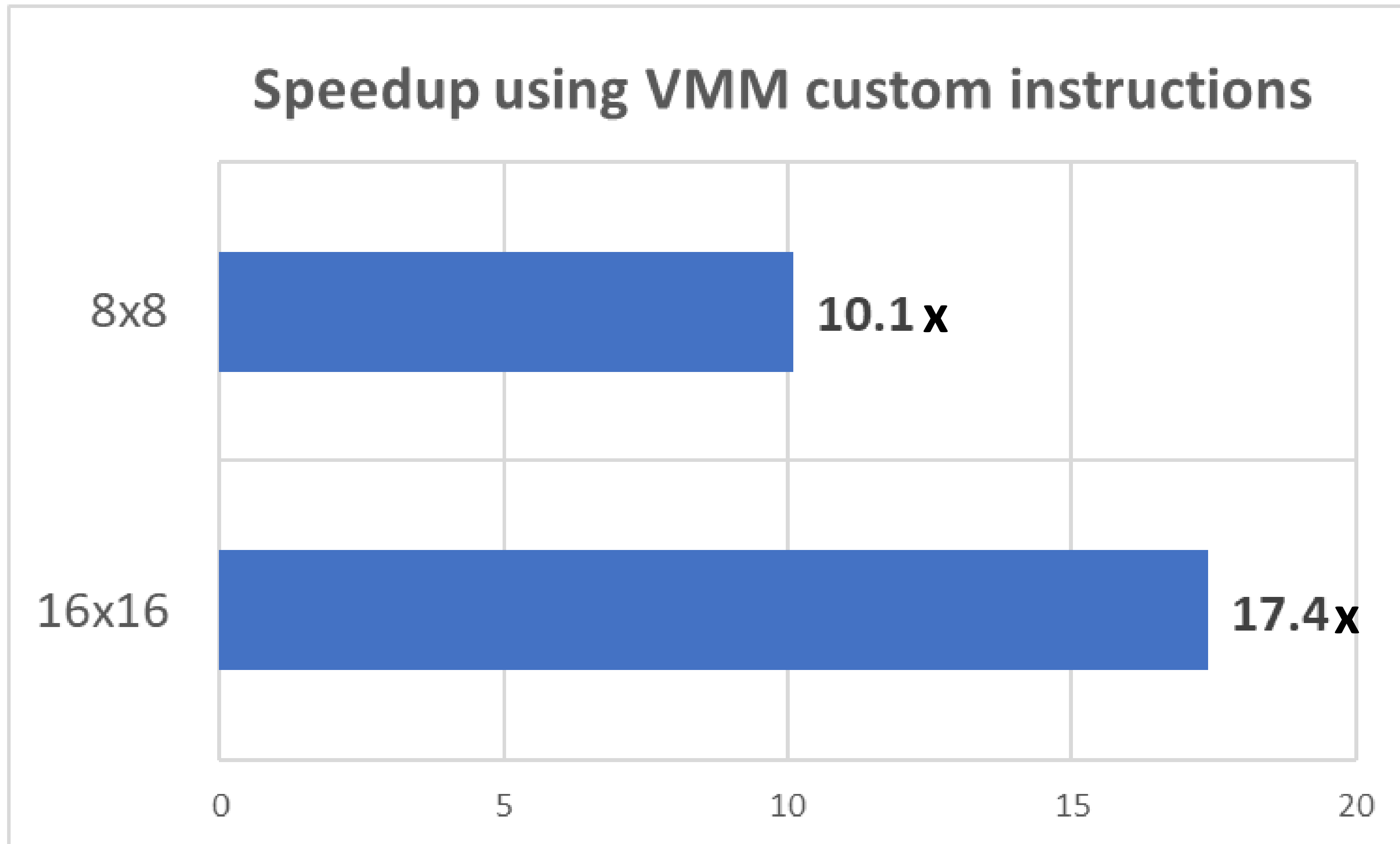


# AI extensions – PIM VMM AFU

- **vmm.ld**
  - Load results from PIM memory
- **vmm.sd**
  - Store weights to PIM memory
- **vmm**
  - Perform in-memory VMM operation
  - 1x8 * 8x8
  - 1x16 * 16x16



# Preliminary Results with PIM VMM Instruction




**Baseline** – Simple C program implementing VMM as nested 'for' loops

Custom VMM instructions exposed to C via compiler intrinsics

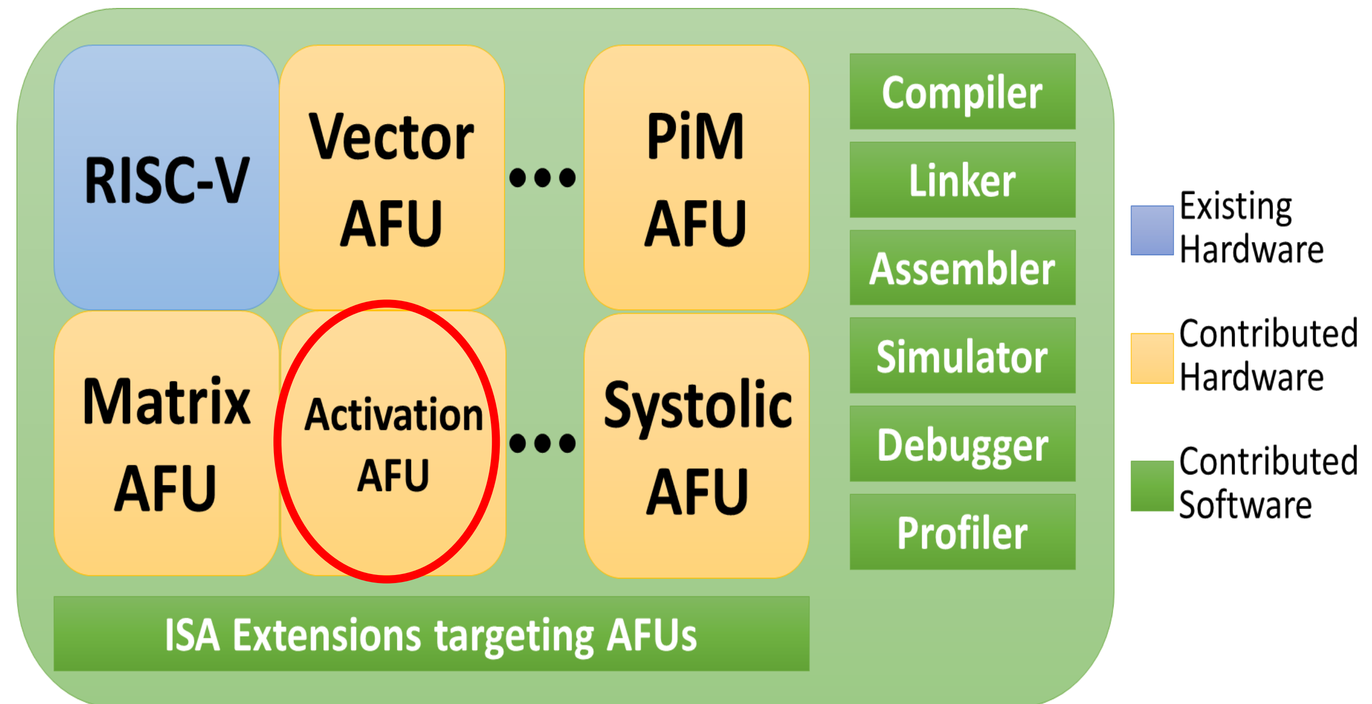
1.6x better performance than writing Assembly

Manuscript under preparation

# Planned AI extensions – Activation AFU

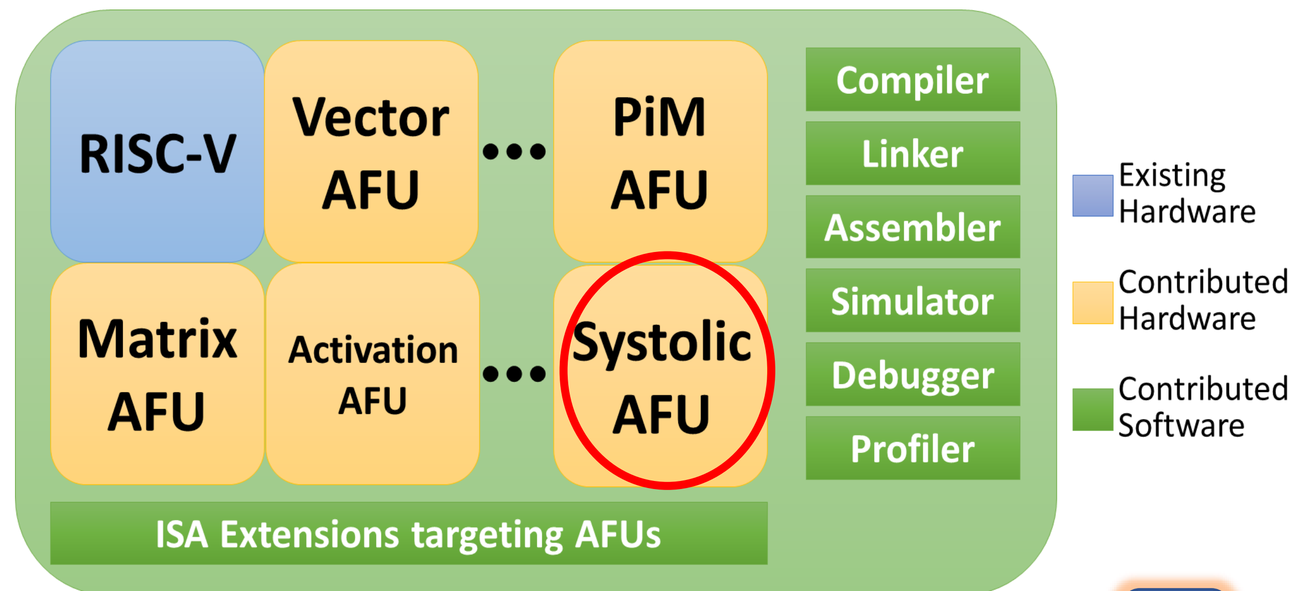
Hardware support for activation functions

- ReLU
- Sigmoid
- Tanh
- Swish



# Planned AI extensions – Systolic AFU

- Load weights into the systolic array
- Store output activations – from systolic array to processor registers/scratchpad
- Prepare input activations – pre-processing instructions
- Systolic MAC
- Flush weights / stop



# 2-step Compiler & Issues

Solved by hardware designers

# Hardware/Software Co-design

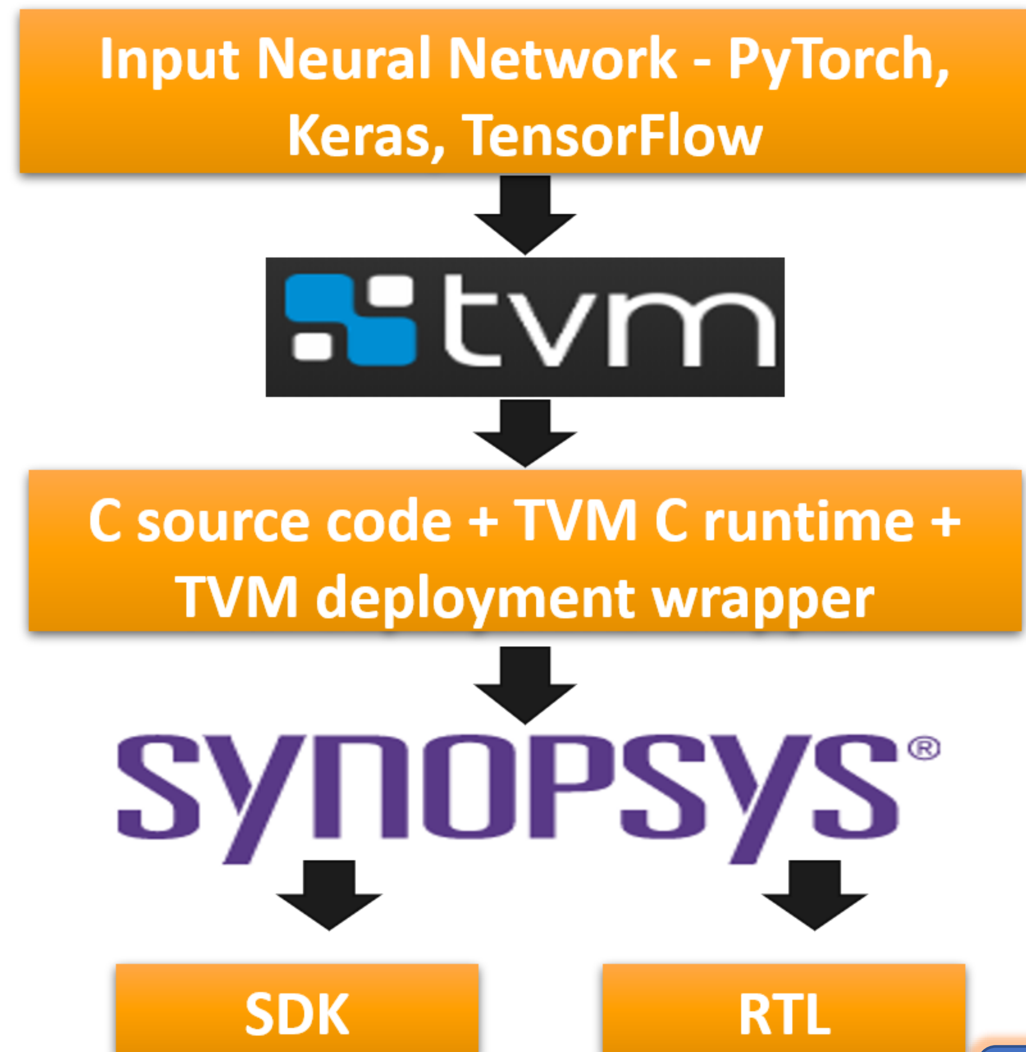


✓End-to-end design methodology
- TVM on the frontend
- Synopsys ASIP Designer on the backend

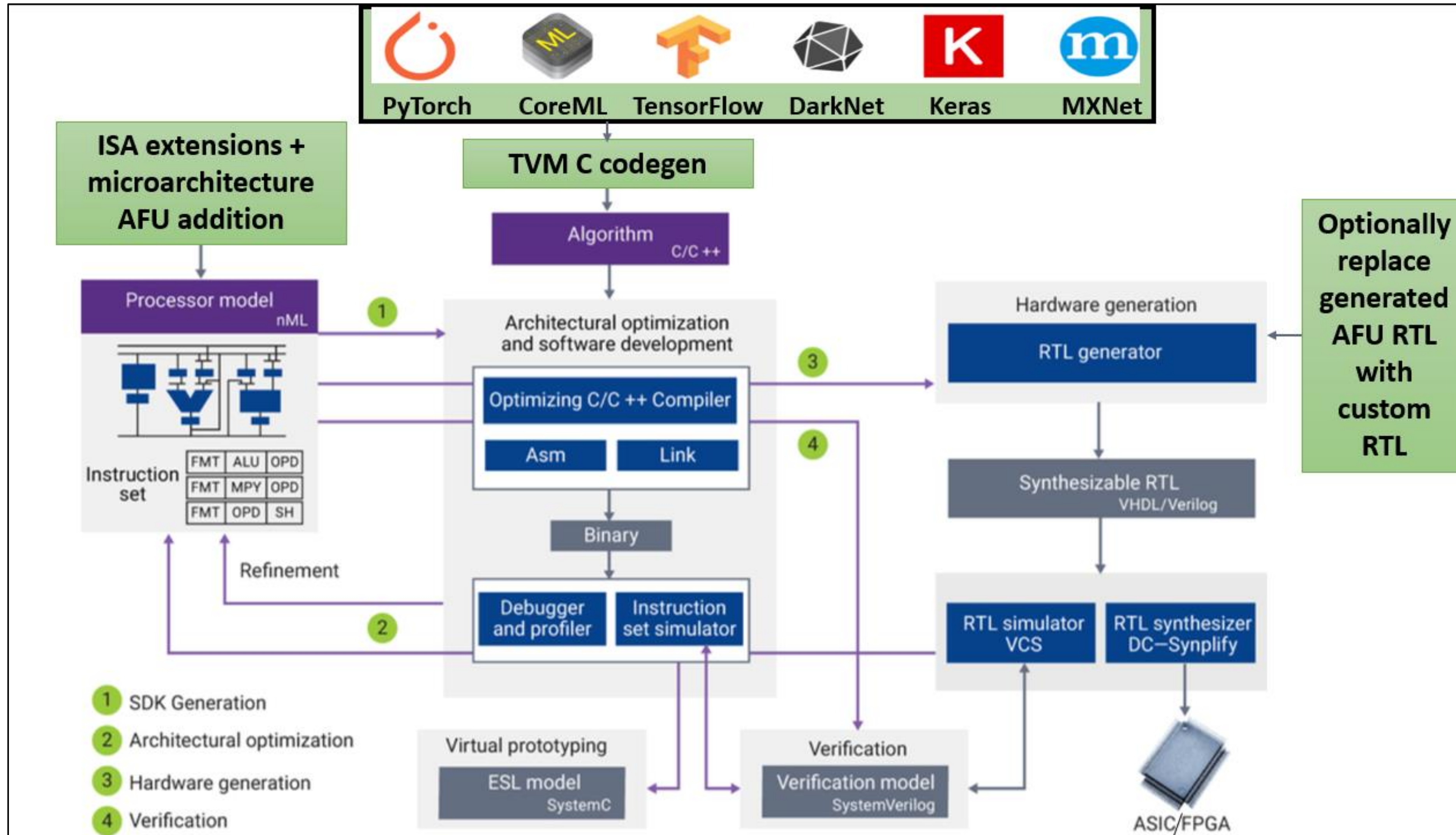
✓Support for multiple Domain Specific Language (DSL) frontends - Pytorch, MXNet, TFLite, Tensorflow, DarkNet etc.

✓Verified with both 32 and 64-bit RISC-V

✓Quantization support



# ASIP Designer Enhancements



# Compilers are hard!

- 2-step compilation – TVM + ASIP Designer generated C compiler
- Goal is to have no/minimum interaction with TVM generated C code

- **Problem 1** – ASIP Compiler is not smart enough to detect opportunities for complex instructions like VMM, GEMM etc.
  - Works well for simple instructions like MAC.
- **Solution** - Expose new instructions to TVM via compiler intrinsics.

Split loops for convolution schedule in TVM

↓

Replace inner loops with PIM VMM instructions

↓

TVM calls AI-RISC VMM instructions in C code

# Compiler issues with custom instructions

- **Problem 2** – Breaking the convolution schedule leads to accuracy issues.
- **Issue 1** – Wrong allocation of operands
  - Solved by TVM buffer and strided access of operands from bigger matrix
- **Issue 2** – Wrong data type in quantized NN
  - Defined input/output data types in TVM hardware intrinsic call
  - 8-bit inputs and 16/32-bit accumulated result.
- **Issue 3** – Kernel and Input data layout
  - TVM supports only a few Input/Filter layouts with specific ISA.
  - Adding support for required Input-Filter layout combinations in TVM for C hardware target used in AI-RISC.

hplp High-Performance Low-Power UNIVERSITY of VIRGINIA

# More Compiler issues

- **Problem 3** – TVM support for breaking the convolution computation to match custom extension kernel size is limited.
  - TVM throws random errors when breaking the computation schedule using "tensorize" schedule pass.
  - Exact same convolution works with tensorize as a standalone kernel but not as a part of neural network.

- **Solution** – Trying to debug the exact issue but till we find a reliable solution we work with what we have.
  - Breaking the computation schedule into error-free parts and adapting the custom instructions accordingly for testing purposes.

# TVM generated C code

```
    for (int32_t k_outer = 0; k_outer < 16; ++k_outer) {
      for (int32_t y_inner = 0; y_inner < 8; ++y_inner) {
        for (int32_t k_inner = 0; k_inner < 8; ++k_inner) {
          ((int32_t*)compute1)[(((z_outer_y_outer_fused * 8) + y_inner))] = (((int32_t*)compute1)[(((z_outer_y_outer_fused * 8) + y_inner)
)] + (((int32_t)((int8_t*)placeholder)[(((k_outer * 8) + k_inner))]) * ((int32_t)((int8_t*)placeholder1)[(((((z_outer_y_outer_fused * 1024
) + (y_inner * 128)) + (k_outer * 8)) + k_inner))])));
        }
      }
    }
  }
```

Without Custom instructions

```
    for (int32_t k_outer = 0; k_outer < 16; ++k_outer) {
      (void)gemm_1x8x8_update_ISQWNLHM(((int8_t *)placeholder + ((k_outer * 8))), ((int8_t *)placeholder1 + (((z_outer_y_outer_fused * 102
4) + (k_outer * 8)))), ((int32_t *)compute1 + ((z_outer_y_outer_fused * 8))), 8, 128, 8);
    }
  }
```

```
void gemm_1x8x8_update_ISQWNLHM(
    int8_t *aa, int8_t *bb, int32_t *cc,
    int A_stride, int B_stride, int C_stride) {

  for (int i = 0; i < 8; i++) {
    PIM_mem_store((char*)i,*((long*)(bb+i*B_stride)));
  }
  chess_memory_fence();

  long out0 = PIM_mac_0(*((long*) (aa)));
  long chess_storage(x13) out1 = PIM_mac_1(*((long*) (aa)));
  long chess_storage(x14) out2 = PIM_mac_2(*((long*) (aa)));
  long out3 = PIM_mac_3(*((long*) (aa)));
  long out[4] = { out0, out1, out2, out3};

  for (int i = 0; i < 8; i++) {
    cc[i] += *((int32_t*)(&out) + i);
  }
}
```

With Custom instructions

# Chess compiler intrinsic

```
promotion    void PIM_mem_store (  char*,  vchar8) = void  PIM_mem_store  (addr,w64);
promotion    void PIM_mem_store (  char*,  int) = void  PIM_mem_store  (addr,w64);
promotion    void PIM_mem_store (  char*,  long) = void  PIM_mem_store  (addr,w64);
promotion    long PIM_mac_0 (long) = w64 PIM_vmm_0  (w64);
promotion    long PIM_mac_1 (long) = w64 PIM_vmm_1  (w64);
promotion    long PIM_mac_2 (long) = w64 PIM_vmm_2  (w64);
promotion    long PIM_mac_3 (long) = w64 PIM_vmm_3  (w64);
```

# Chess_view rule

```
chess_view() {
PIM_mem[pim_addr] = pim_wr;
} -> {
PIM_mem_store(pim_addr,pim_wr);}

chess_view() {
pim_rd_mac`EX` = PIM_mac[pim_addr=0`ID`]`EX`;
pimOUT0`EX` = pimmac (pimIN`EX`, pim_rd_mac`EX`, pimOUT1`EX`, pimOUT2`EX`, pimOUT3`EX`);
} -> {
pimOUT0`EX` = PIM_vmm_0(pimIN`EX`);
pimOUT1`EX` = PIM_vmm_1(pimIN`EX`);
pimOUT2`EX` = PIM_vmm_2(pimIN`EX`);
pimOUT3`EX` = PIM_vmm_3(pimIN`EX`);}
```

# Instruction definition nML

```
opn pim_rr_instr(rdh: eX, rdl: eX, rs1: eX)
{
  action {
  stage ID..EX:
                        pim_rd_mac`EX` = PIM_mac[pim_addr=0`ID`]`EX`;
  stage ID:             pid_S1 = r1 = X[rs1];
  stage EX:             pimIN = pid_S1;
                        //pimmac (pimIN, pim_rd_mac, pimOUTl, pimOUTh) @pim64x64;
                        pimOUT0 = pimmac (pimIN, pim_rd_mac, pimOUT1, pimOUT2, pimOUT3) @pim64x64
                        pex_D1 = tex_D1 = pimOUT0;
                        pex_D2 = tex_D2 = pimOUT1;
                        pex_D3 = tex_D3 = pimOUT2;
                        pex_D4 = tex_D4 = pimOUT3;
    stage ME:           pme_D1 = tme_D1 = pex_D1;
                        pme_D2 = tme_D2 = pex_D2;
                        pme_D3 = tme_D3 = pex_D3;
                        pme_D4 = tme_D4 = pex_D4;
    stage WB:
      if (rdl: x0)      { w1_dead = w1 = pme_D1;
                          w2_dead = w2 = pme_D2;
                          w3_dead = w3 = pme_D3;
                          w4_dead = w4 = pme_D4;}
      else              { X[rdl] = w1 = pme_D1;
                          X[13] = w2 = pme_D2;
                          X[14] = w3 = pme_D3;
                          X[rdh] = w4 = pme_D4;}
  }
  syntax : "vmm"     PADMNM " " rdh "," PADMNM " " rdl "," PADOP1 rs1 ;
  image  : "0000000"::rdh::rs1::"001"::rdl, class(pim_rrr);
}
```

# Compiled assembly with new instructions

```
6970    0x01 0x50 0x30 0x5b  vmm.sd   x21, 0(x0)
6974    0x01 0x40 0x30 0xdb  vmm.sd   x20, 1(x0)
6978    0x08 0x01 0xba 0x0b  ld       x20, 128(x3!)
6982    0x01 0x40 0x31 0x5b  vmm.sd   x20, 2(x0)
6986    0x08 0x01 0xba 0x0b  ld       x20, 128(x3!)
6990    0x01 0x40 0x31 0xdb  vmm.sd   x20, 3(x0)
6994    0x08 0x01 0xba 0x0b  ld       x20, 128(x3!)
6998    0x01 0x40 0x32 0x5b  vmm.sd   x20, 4(x0)
7002    0x08 0x01 0xba 0x0b  ld       x20, 128(x3!)
7006    0x01 0x40 0x32 0xdb  vmm.sd   x20, 5(x0)
7010    0x08 0x01 0xba 0x0b  ld       x20, 128(x3!)
7014    0x01 0x40 0x33 0x5b  vmm.sd   x20, 6(x0)
7018    0xc8 0x81 0xba 0x0b  ld       x20, -888(x3!)
7022    0x01 0x40 0x33 0xdb  vmm.sd   x20, 7(x0)
7026    0x00 0x83 0x3a 0x0b  ld       x20, 8(x6!)
7030    0x41 0x1c            c.lw     x15, 0(x10)
7032    0x00 0xca 0x15 0xdb  vmm      x12,  x11, x20
7036    0x00 0x05 0x84 0x1b  sext.w   x8,  x11
```

vv8dn@virginia.edu

# Emulation/Definition of AFU

```
w64 pimmac(w64 a, v8w64 pim_rd_mac, w64 & out1, w64 & out2, w64 & out3)  {
    v8w32 r = 0;
    v8w08 aa = a;
    w64 out0;
    for (int i = 0; i < 8; i++){
      for (int j = 0; j < 8; j++){
          v8w08 temp = pim_rd_mac[j];
          int32_t inter = 0;
          inter = aa[j] * temp[i];
          r[i] += inter;
      }
    }
    out0 = r[1]::r[0];
    out1 = r[3]::r[2];
    out2 = r[5]::r[4];
    out3 = r[7]::r[6];
    return out0;
}
```

# Instructions through instruction viewer

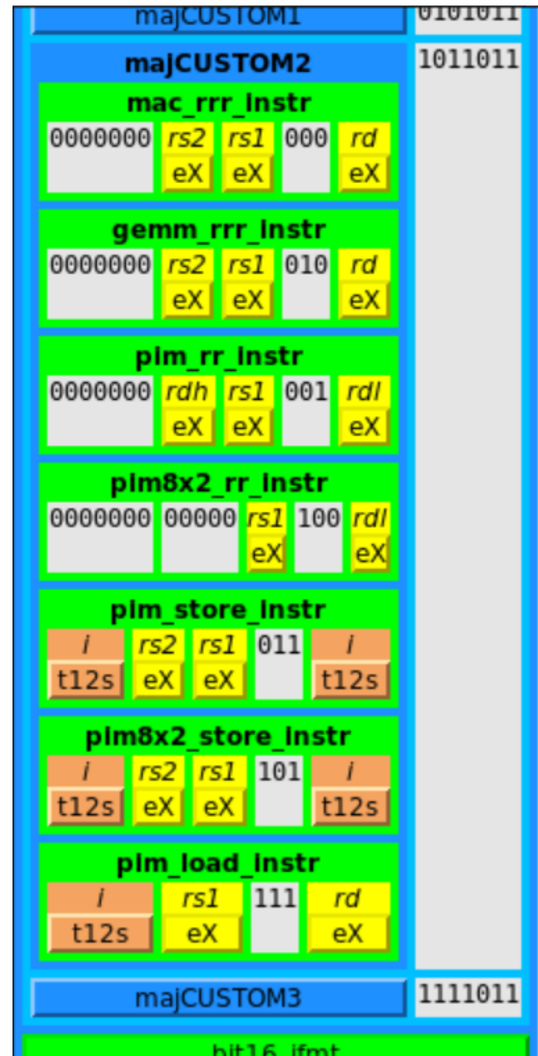
```
3                   2                   1
10987654321098765432109876543210
```

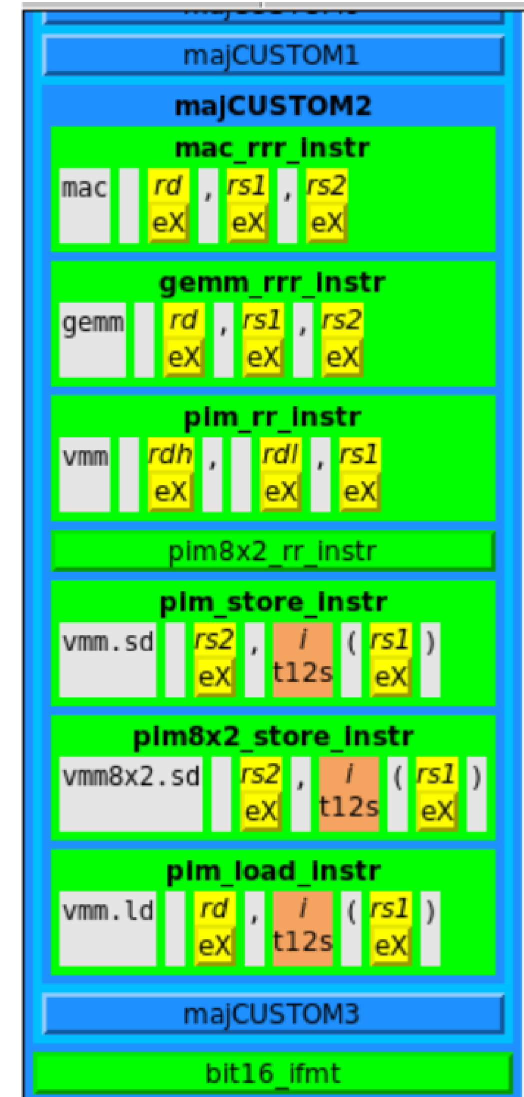
**trv64p5x**

**bit32_ifmt**

| Major opcode | Encoding |
|---|---|
| majOP | 0110011 |
| majOP_IMM | 0010011 |
| majLOAD | 0000011 |
| majSTORE | 0100011 |
| majBRANCH | 1100011 |
| majJAL | 1101111 |
| majJALR | 1100111 |
| majLUI | 0110111 |
| majAUIPC | 0010111 |
| majOP_32 | 0111011 |
| majOP_IMM_32 | 0011011 |
| majCUSTOM0 | 0001011 |
| majCUSTOM1 | 0101011 |
| **majCUSTOM2** | 1011011 |

```
mac_rrr_instr
0000000 rs2 : eX rs1 : eX 000 rd : eX
gemm_rrr_instr
0000000 rs2 : eX rs1 : eX 010 rd : eX
pim_rr_instr
0000000 rdh : eX rs1 : eX 001 rdl : eX
pim8x2_rr_instr
0000000 00000 rs1 : eX 100 rdl : eX
pim_store_instr
i : t12s rs2 : eX rs1 : eX 011 i : t12s
pim8x2_store_instr
i : t12s rs2 : eX rs1 : eX 101 i : t12s
pim_load_instr
i : t12s rs1 : eX 111 rd : eX
```




vv8dn@virginia.edu

# Results

Performance improvements with AI-RISC

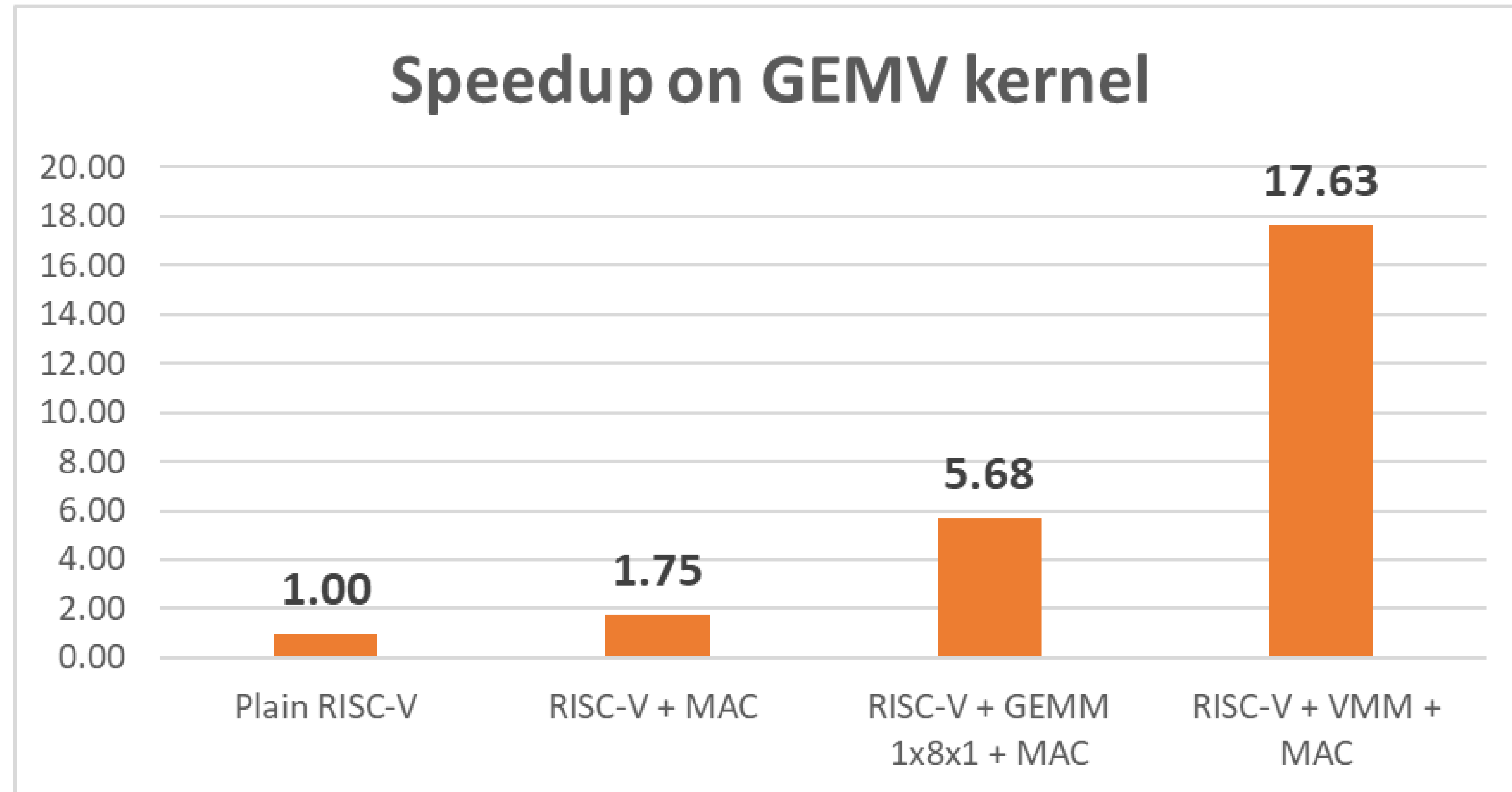
# Evaluation Methodology

- Benchmark
  - ResNet-8 from TinyMLPerf
  - GEMV kernel
- Baseline
  - 5-stage in-order 64-bit RISC-V RV64IMC
- Compilation
  - TVM (TFLite to C) + Custom C compiler (C to binary)
- Simulation Framework
  - Cycle-accurate Simulator

vv8dn@virginia.edu

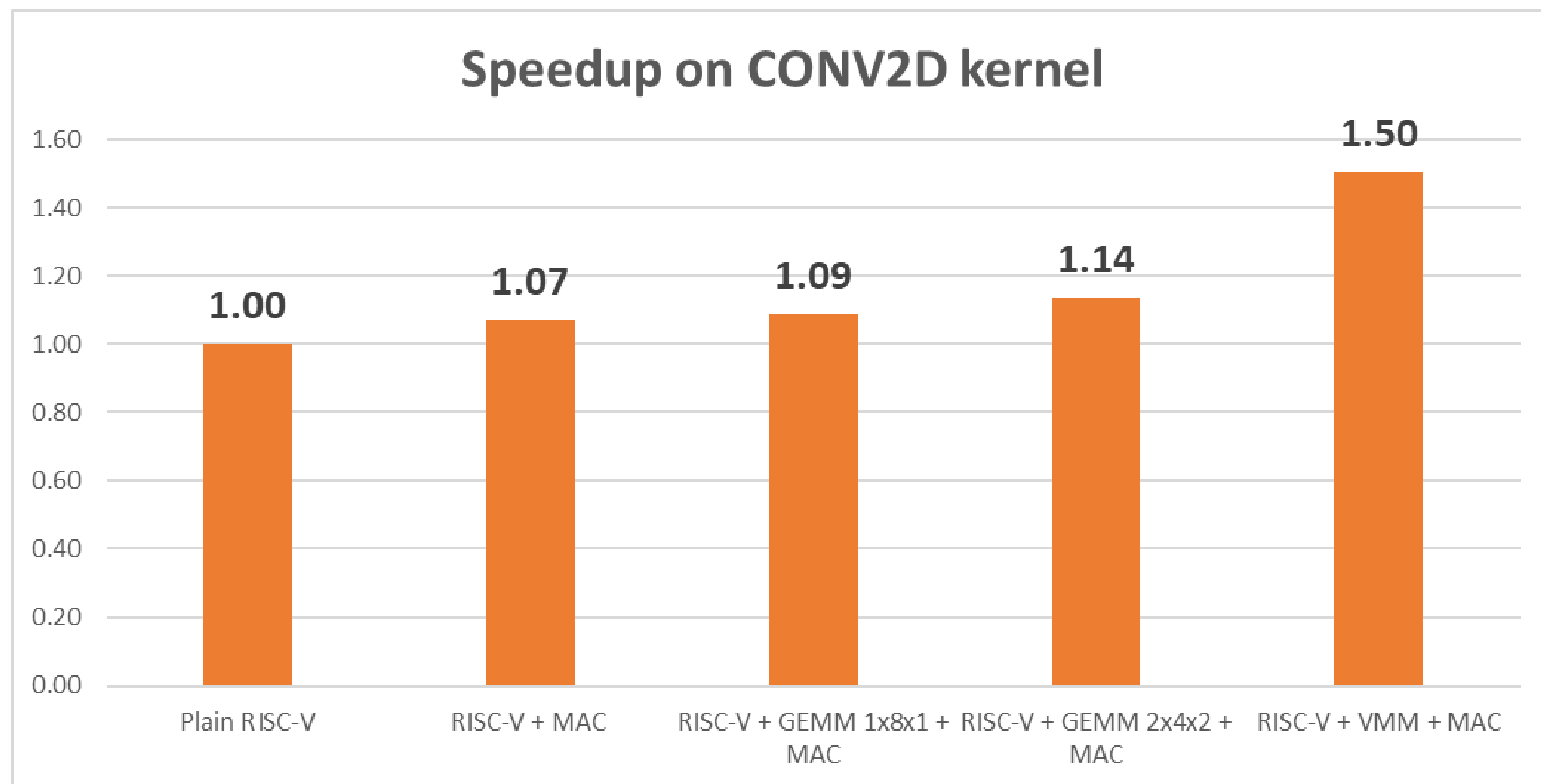
# Speedup on GEMV kernel

- A Matrix → 8x8
- B Vector → 1x8
- Input datatype → int8
- Output datatype → int16



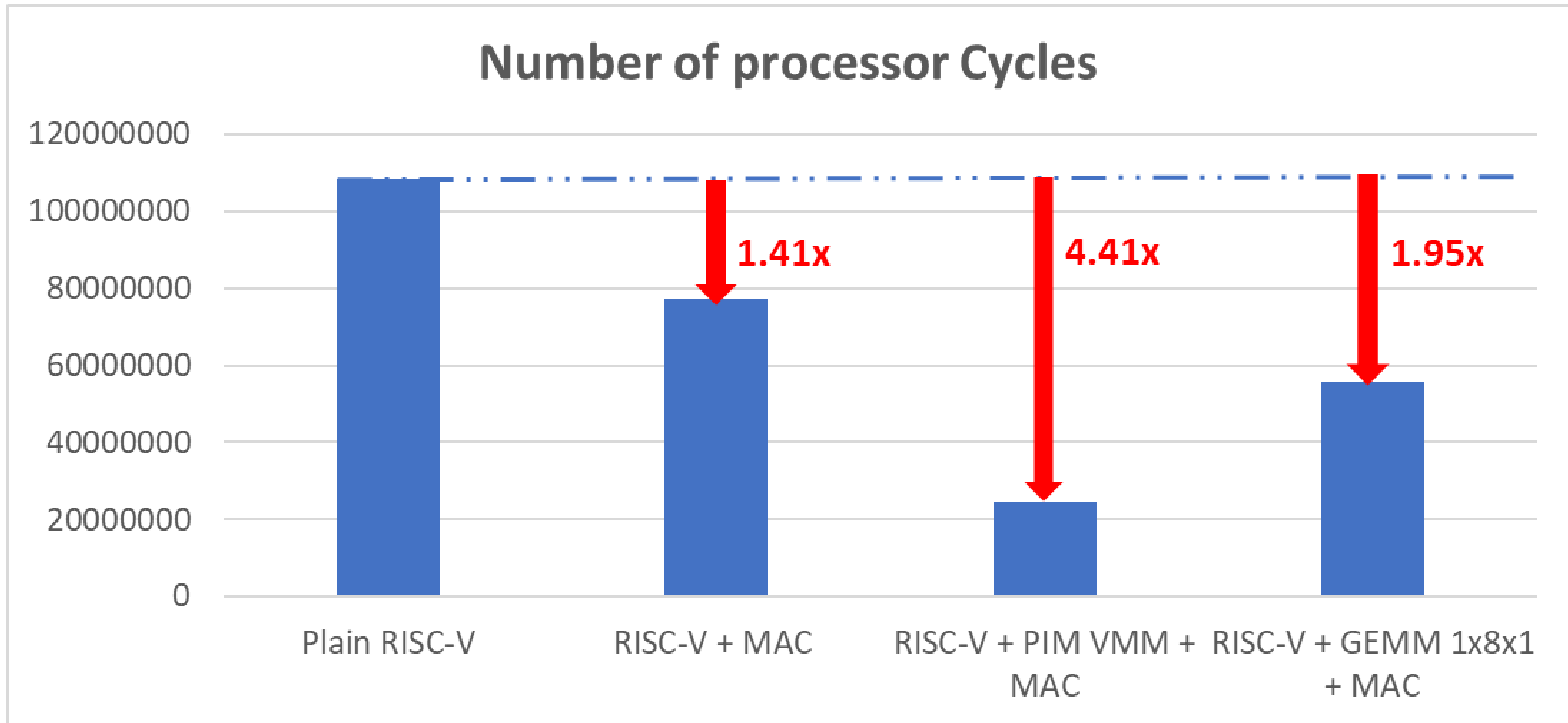
# Speedup on single CONV2D kernel

- Input image → 7x7
- Input channels → 8
- Filter → 2x2
- Output channels → 2
- Input datatype → int8
- Output datatype → int16
- data_layout → NHWC
- kernel_layout → HWIO

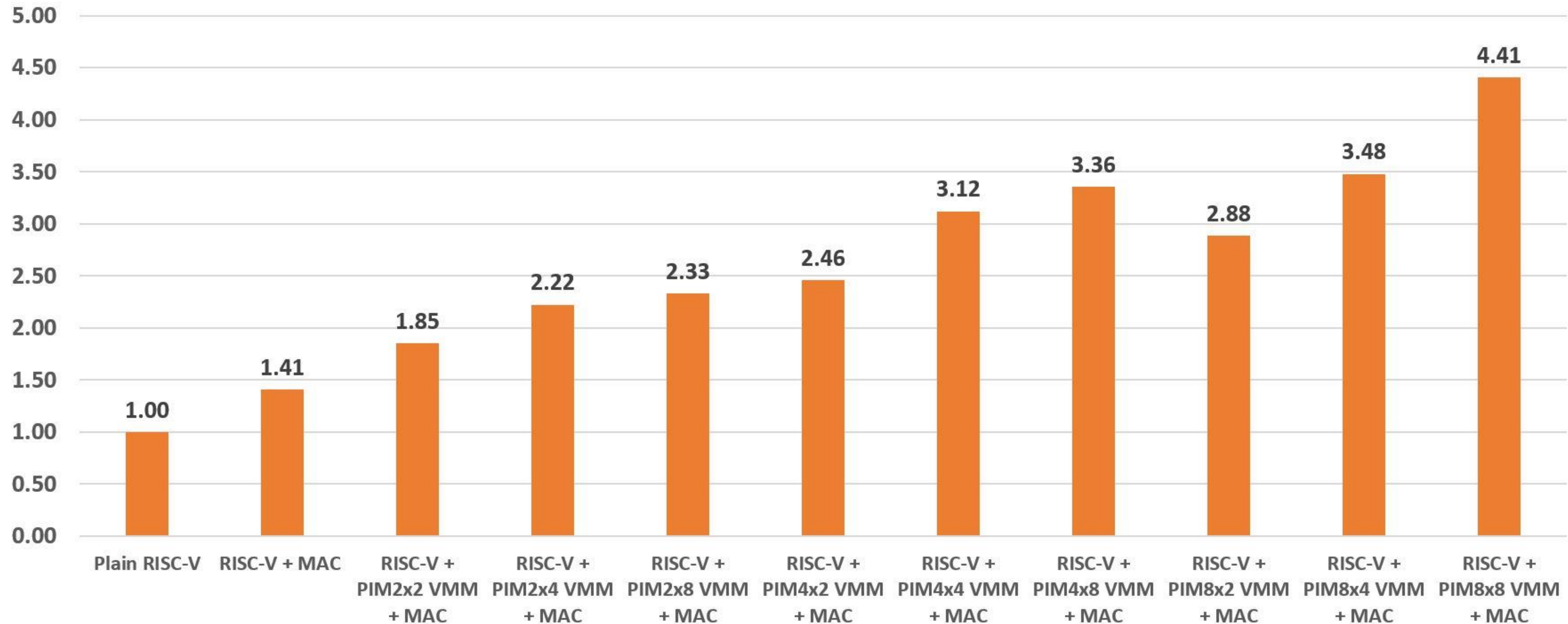



vv8dn@virginia.edu

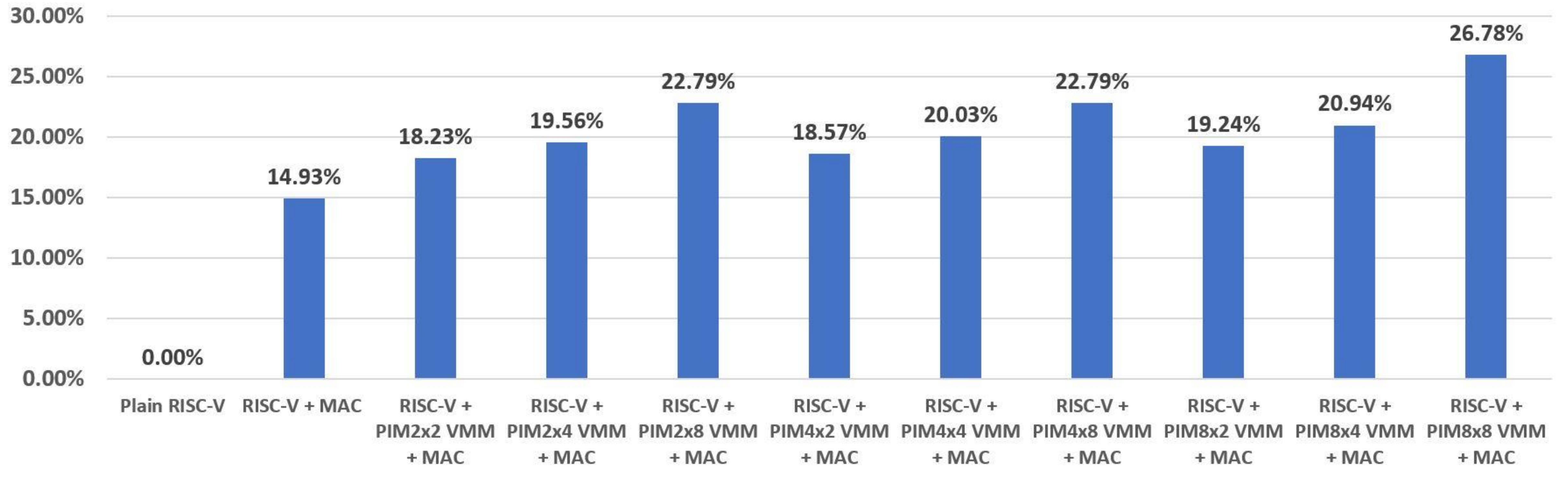
# Speedup on ResNet-8 network from TinyMLPerf



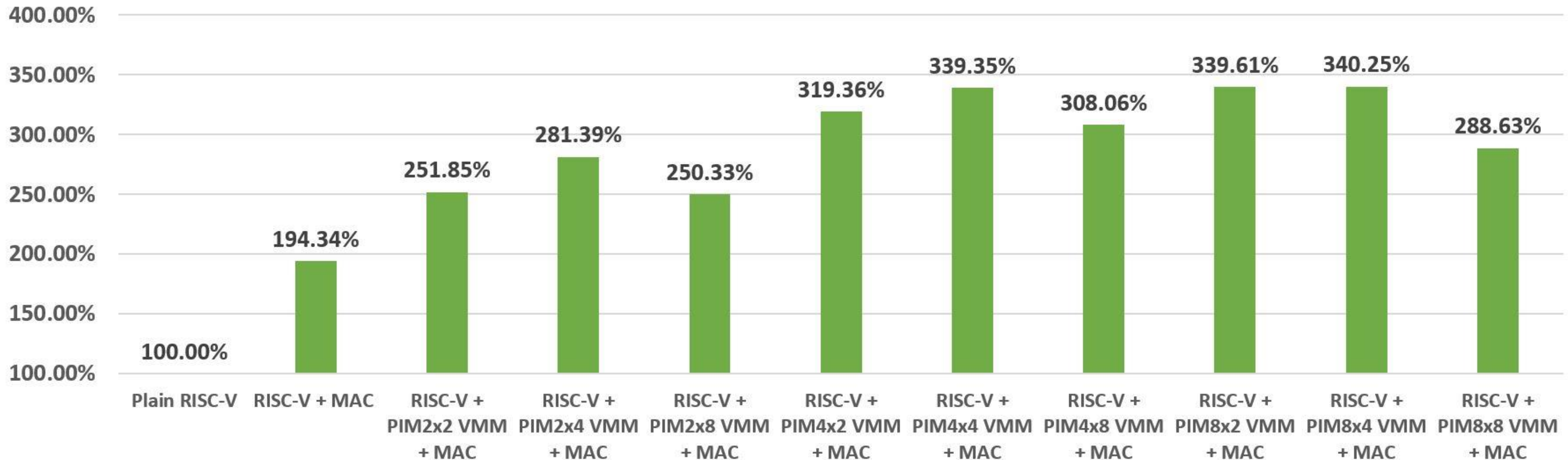
# Design-Space Exploration for PIM VMM



# Area Overhead



# FoM for optimal PIM size



# Vector "V" extension addition to AI-RISC

- Undergrad project by Nate Hunter, Noah Mills, Greg Vavoso and Bill Yang.
- Subset of Vector extension instructions added to AI-RISC:
  - Addition and Subtraction
  - Logical AND/OR/XOR/MIN/MAX
  - Shift and Move
  - Multiplication

Speedup Examples

| Example | Vector instructions | Scalar processor cycles | Vector processor cycles | Speedup |
|---|---|---|---|---|
| Subtract on 128-byte vectors | vsub.vv | 128 | 1 | **128** |
| Dot product on 32-word vectors | vmul.vv, vredsum.vs | 33 + 31 = 64 | 2 + 5 = 7 | **9.1** |

# Summary

- AI-RISC = Custom RISC-V processor for Edge AI
- Tightly integrated AI Functional Units (AFU)
- Custom ISA extensions to RISC-V
- Complete SDK generation including compiler support for PyTorch, TensorFlow etc.
- Speedup compared to RV64IMC → 17.6x for GEMV, 4.4x for ResNet-8
- Scalable, flexible and support for both AI and non-AI applications

# Publications

- V. Verma, M. Stan, “AI-RISC - Custom Extensions to RISC-V for Energy-efficient AI Inference at the Edge of IoT,” **RISC-V Summit** co-located with Design Automation Conference (RISC-V), San Francisco, CA, December 2021.
- V. Verma, T. Tracy, M. Stan, “EXTREM-EDGE - EXtensions To RISC-V for Energy-efficient ML inference at the EDGE of IoT,” The 12th **International Green and Sustainable Computing Conference (IGSC)**, October 2021.
- V. Verma, M. Stan, “AI-RISC: Extending RISC-V with tightly integrated accelerators and custom instructions for AI inference at the Edge of IoT,” IBM IEEE CAS/EDS - **AI Compute Symposium (AICS)**,October 2021.
- V. Verma, M. Stan, “AI-RISC: Scalable open source processor for AI applications at edge of IoT,” **Design Automation Conference Young Student Fellow Program poster session (DAC YFP), July 2020.** ***(Best Poster Award)***

# Questions?

- This work is funded by SRC under GRC AIHW task 2945.001.

vv8dn@virginia.edu