Leveraging TVM as a front-end compiler for RISC-V based custom tinyML processor

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

- **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
But tinyML/Edge AI is different!

- Smaller neural network models
- Smaller batch sizes ($\approx 1$)
- Edge devices are power, cost, area and size limited
- Edge processors 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

RISC-V

Vector AFU

PiM AFU

Matrix AFU

Activation AFU

Systolic AFU

Compiler

Linker

Assembler

Simulator

Debugger

Profiler

ISA Extensions targeting AFUs

Existing Hardware

Contributed Hardware

Contributed Software
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
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.
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

```c
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_outter_y_outter_fused * 8) + y_inner)] = (((int32_t*)compute1)[((z_outter_y_outter_fused * 8) + y_inner)] + (((int32_t*)(int8_t*)placeholder)[((k_outer * 8) + k_inner)]) * (((int32_t*)(int8_t*)placeholder1)[(((z_outter_y_outter_fused * 1024) + (y_inner * 128)) + (k_outer * 8)] + k_inner)]);
        }
    }
}

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_outter_y_outter_fused * 1024) + (k_outer * 8))) + (((int32_t*)compute1 + (z_outter_y_outter_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*)1, *((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);
    }
}
```

Without Custom instructions

With Custom instructions

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 $\rightarrow 7\times 7$
- Input channels $\rightarrow 8$
- Filter $\rightarrow 2\times 2$
- Output channels $\rightarrow 2$
- Input datatype $\rightarrow \text{int}8$
- Output datatype $\rightarrow \text{int}16$
- data_layout $\rightarrow \text{NHWC}$
- kernel_layout $\rightarrow \text{HWIO}$

![Speedup on CONV2D kernel](image)
Speedup on ResNet-8 network from MLPerf Tiny

- Plain RISC-V (R): 120,000,000 cycles
- R + MAC (M): 80,000,000 cycles, 1.41x speedup
- R + M + PIM8x8 VMM: 50,000,000 cycles, 4.41x speedup
- R + M + GEMM 1x8x1: 80,000,000 cycles, 1.95x speedup
- Arm Cortex-A72: 80,000,000 cycles, 2.23x speedup

These results indicate significant improvements in processing speed with various optimizations.
Questions?

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