UNIVERSITY OF Southampton

Expanding a RISC-V Processor with Vector Instructions for Accelerating Machine Learning

Presented by Matthew Johns, Harry Cooper, Pete Alexander

Aaryaman Bhattacharya, Byron Theobald, John Holden

Supported by Embecosm

UNIVERSITY OF Southampton

Overview

- Goals
- RISC-V Vector Extension (RVV) and vector algorithms
- Vector accelerator and integration
- Benchmarking, TFL Micro and TinyMLPerf
- Results
- Conclusion

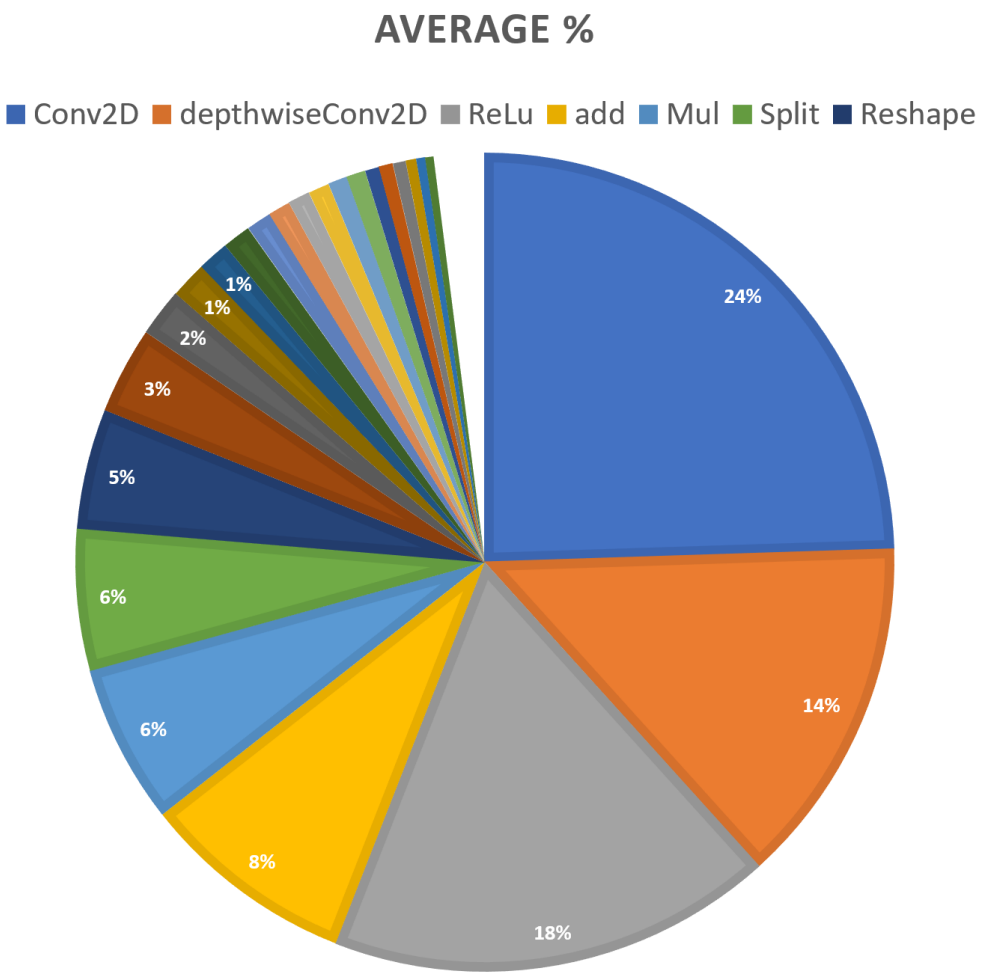
UNIVERSITY OF Southampton

Goals

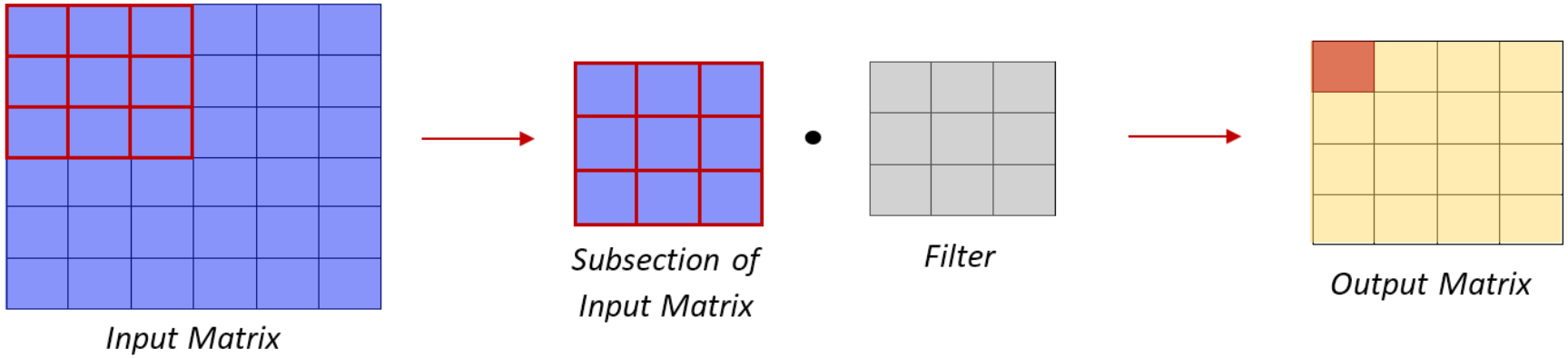
- 6 fourth-year Electronic Engineering students, 10 weeks
- Instruction extension for machine learning - NN inference
- Expand CV32E40P CPU (OpenHW Group)

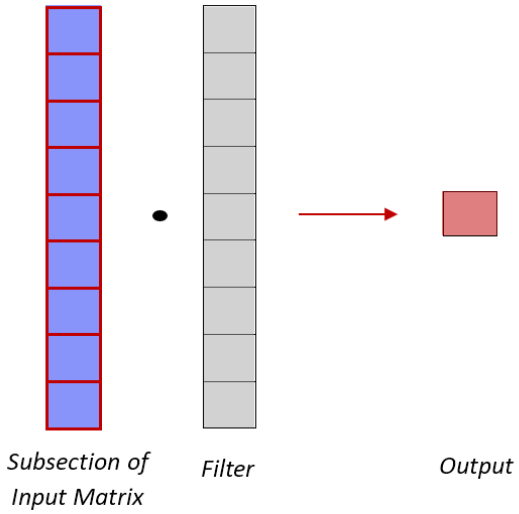
- TinyML
- Open source

Commonly Used Neural Network Functions



Convolution

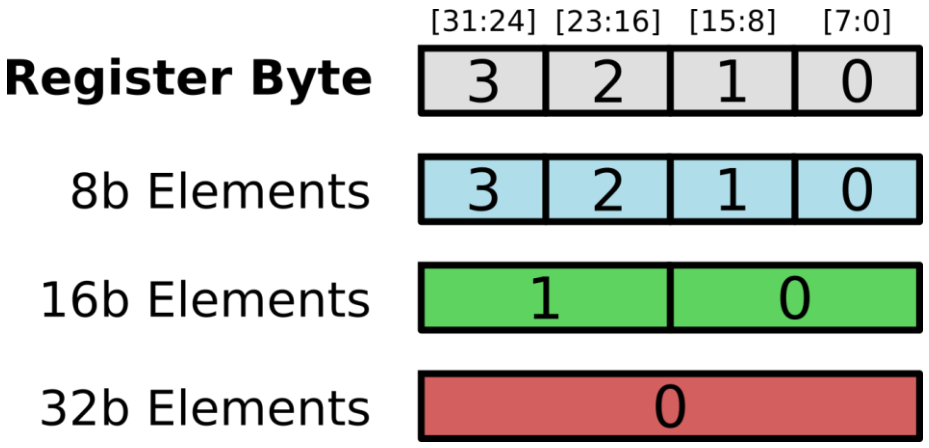




UNIVERSITY OF Southampton

RISC-V Vector Extension

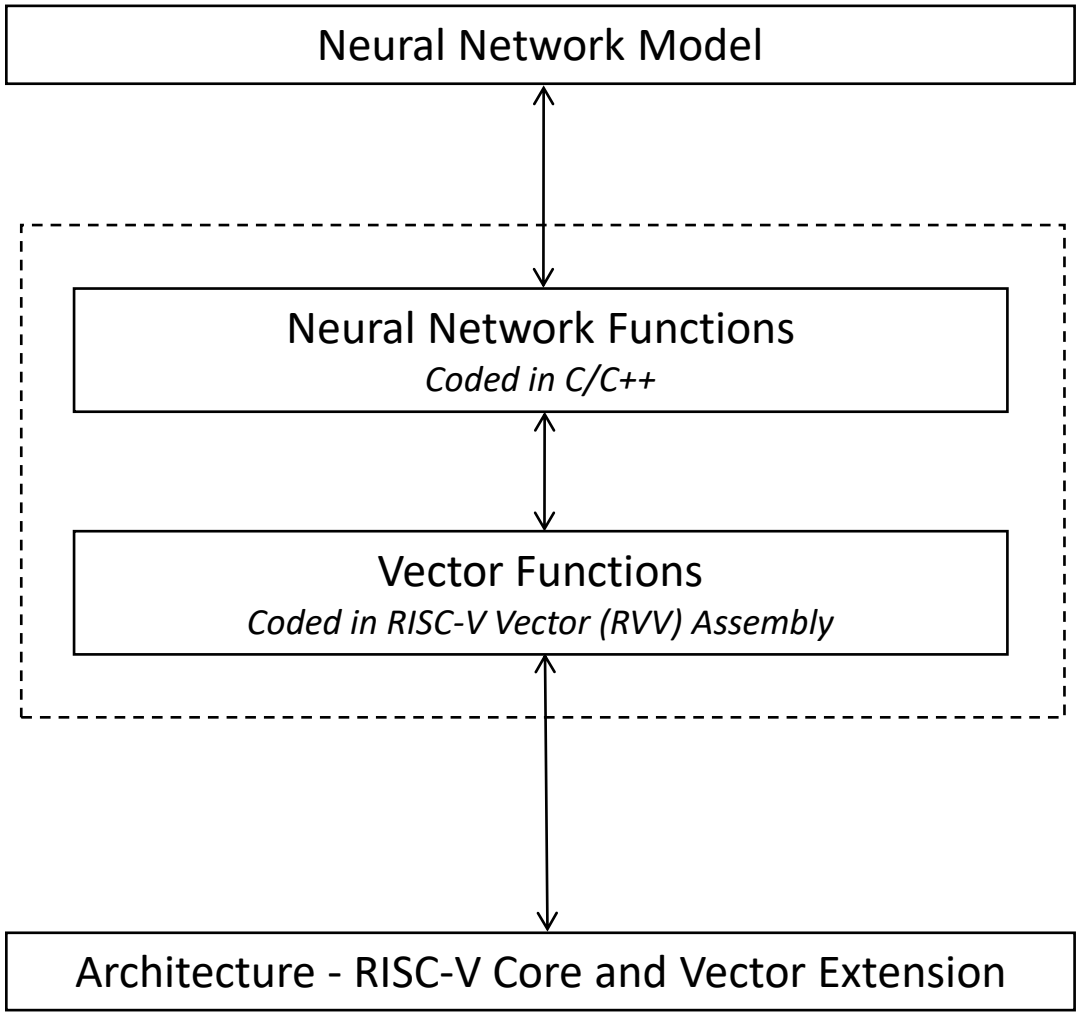
- Support for RVV (v0.8) assembly instructions in RISC-V GNU toolchain
- Supports various element widths in Vector Registers
- Code independent from vector hardware implementation



32-bit Vector Register

UNIVERSITY OF Southampton

Software Design



Vector Function

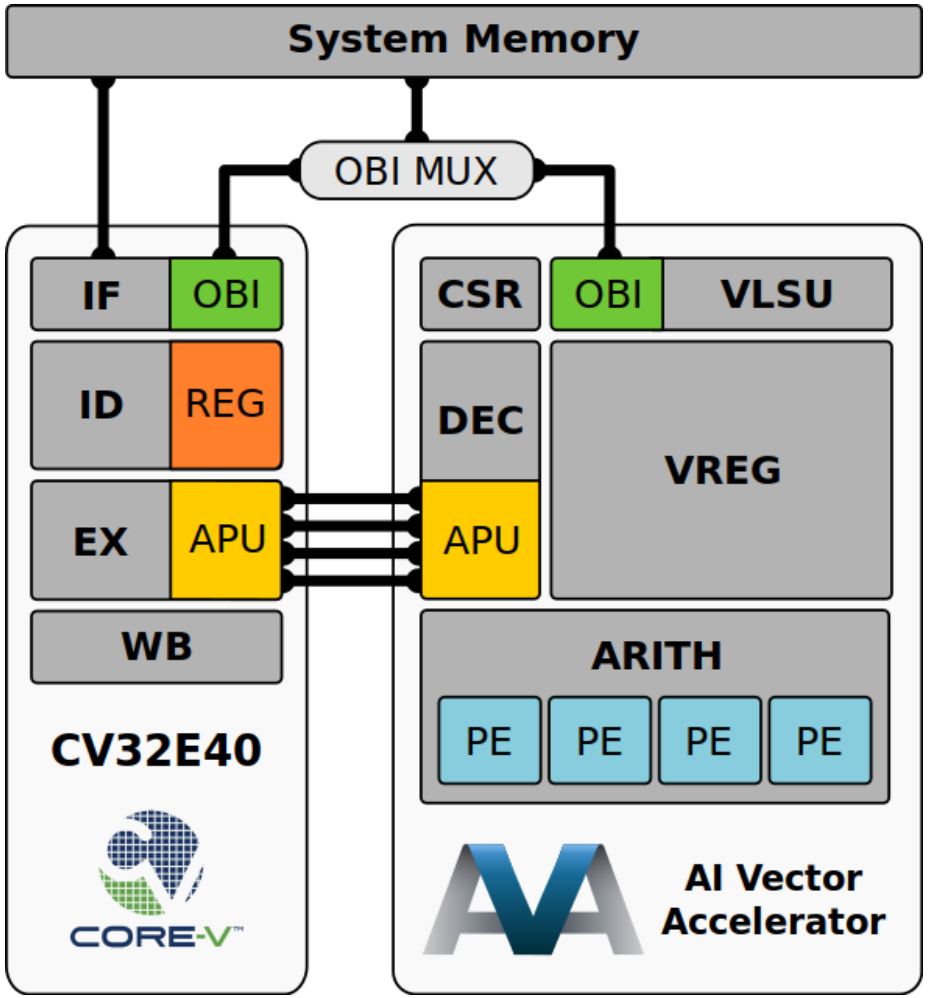
```
vector_add:
vsetvli t0, a0, e8          # Set vl based on 8-bit vectors
vle.v v1, (a1)              # Get first vector
  add a1, a1, t0            # Bump pointer
vle.v v2, (a2)              # Get second vector
  add a2, a2, t0            # Bump pointer
vadd.vv v3, v1, v2          # Sum vectors
vse.v v3, (a3)              # Store result
  add a3, a3, t0            # Bump pointer
  sub a0, a0, t0            # Decrement number done
  bnez a0, vector_add       # Loop back
  ret                       # Finished

# a0 = length of vector(N), a1 = *vector1, a2 = *vector2, a3 = *vectorOut
```

Included Vector Assembly Instructions

Vector Assembly Instruction
vsetvli
vle.v
vlse.v
vse.v
vmv.v.i
vmv.v.x
vmv.x.s
vsadd.vv
vadd.vv
vmul.vv/vx
vwmul.vv/vx
vmax.vx
vmin.vx
vredmax.vs
vwredsum.vs

https://github.com/riscv/riscv-v-spec

Accelerator Design



UNIVERSITY OF Southampton

Accelerator Design - Challenges

- Accelerator integration requires heavy CPU modification
- Complex verification
  - RVV has lots of settings

Benchmarking - TinyMLPerf

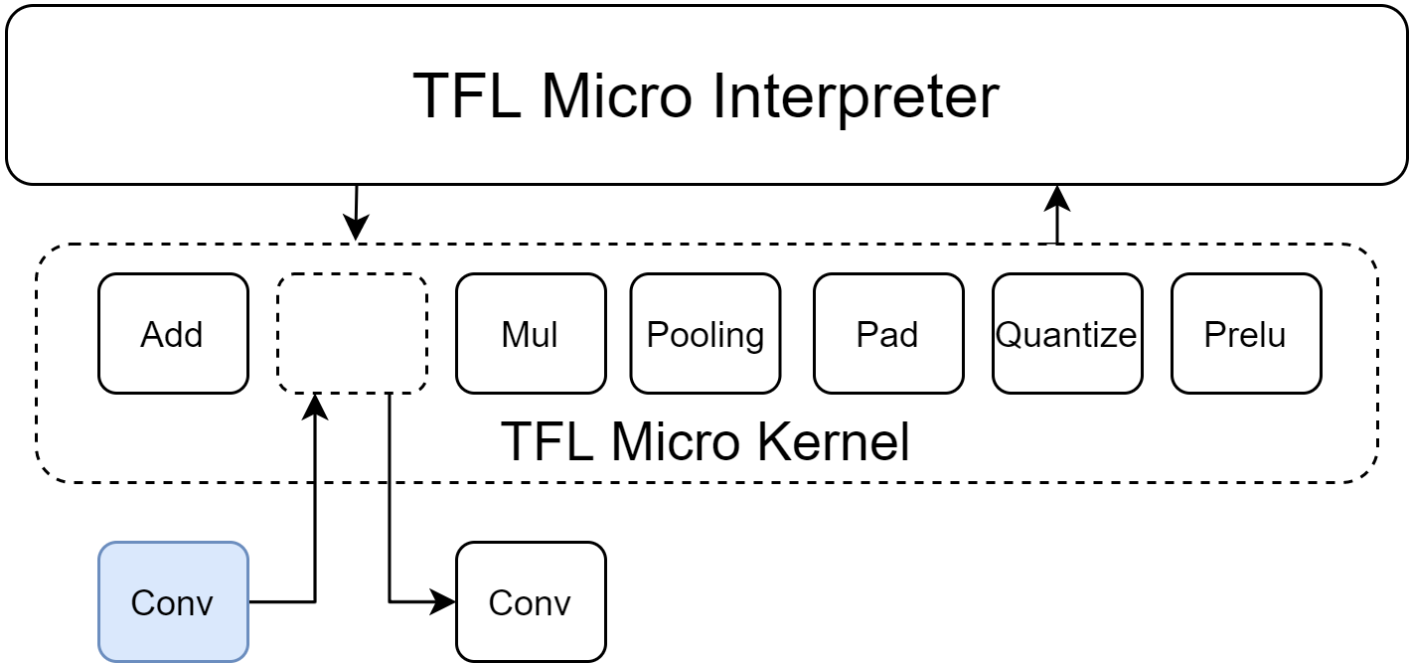
- Benchmarking suite for embedded systems
- Maintained by MLCommons, previously MLPerf
- Benchmarks:
  - Visual Wake Word (VWW)
  - Audio Wake Word
  - Anomaly Detection
- VWW based on TFL Micro
- Links:
  - mlcommons.org
  - github.com/mlcommons/tiny
  - White Paper: arxiv.org/pdf/2003.04821.pdf

UNIVERSITY OF Southampton

Benchmarking - TFL Micro

- Google's cross-platform ML toolkit for embedded devices
- Runs TF models compressed for mobile platforms
- Reimplementation of TensorFlow Lite:
  - Static Memory Allocation
  - Smaller Binary Size

Benchmarking - TFL Micro



Benchmarking - TinyMLPerf CV32E40P Port

UNIVERSITY OF Southampton

- TFL Micro
  - Largely build system edits
  - Used existing RISC-V build infrastructure
  - Hello World example used in tests

- TinyMLPerf
  - Swap official TFL Micro repo with ported fork
  - Add multi-platform build system
  - Add cycle measurement using `mcycle` CSR

UNIVERSITY OF Southampton

Benchmarking - TFL Micro Optimisation

- Not as hard as we feared
- Useful Resource:

  TinyML - Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers

  Pete Warden, Daniel Situnayake
- Many thanks to Pete Warden, Tim Callahan, Tim Ansell for their help

Benchmarking - TFL Micro Optimisation

UNIVERSITY OF Southampton

Reference Implementation

```
tensorflow/lite/micro/kernels
├── add.cc
├── ceil.cc
├── conv.cc
├── depthwise_conv.cc
├── dequantize.cc
└── ...
```

Optimisations Added

```
tensorflow/lite/micro/kernels
├── add.cc
├── ceil.cc
├── conv.cc
├── cv32e40p-ml
│   ├── conv.cc
│   ├── depthwise_conv.cc
│   └── ...
├── depthwise_conv.cc
├── dequantize.cc
└── ...
```

Makefile option for optimisations

```
make … TAGS=cv32e40p-ml
```

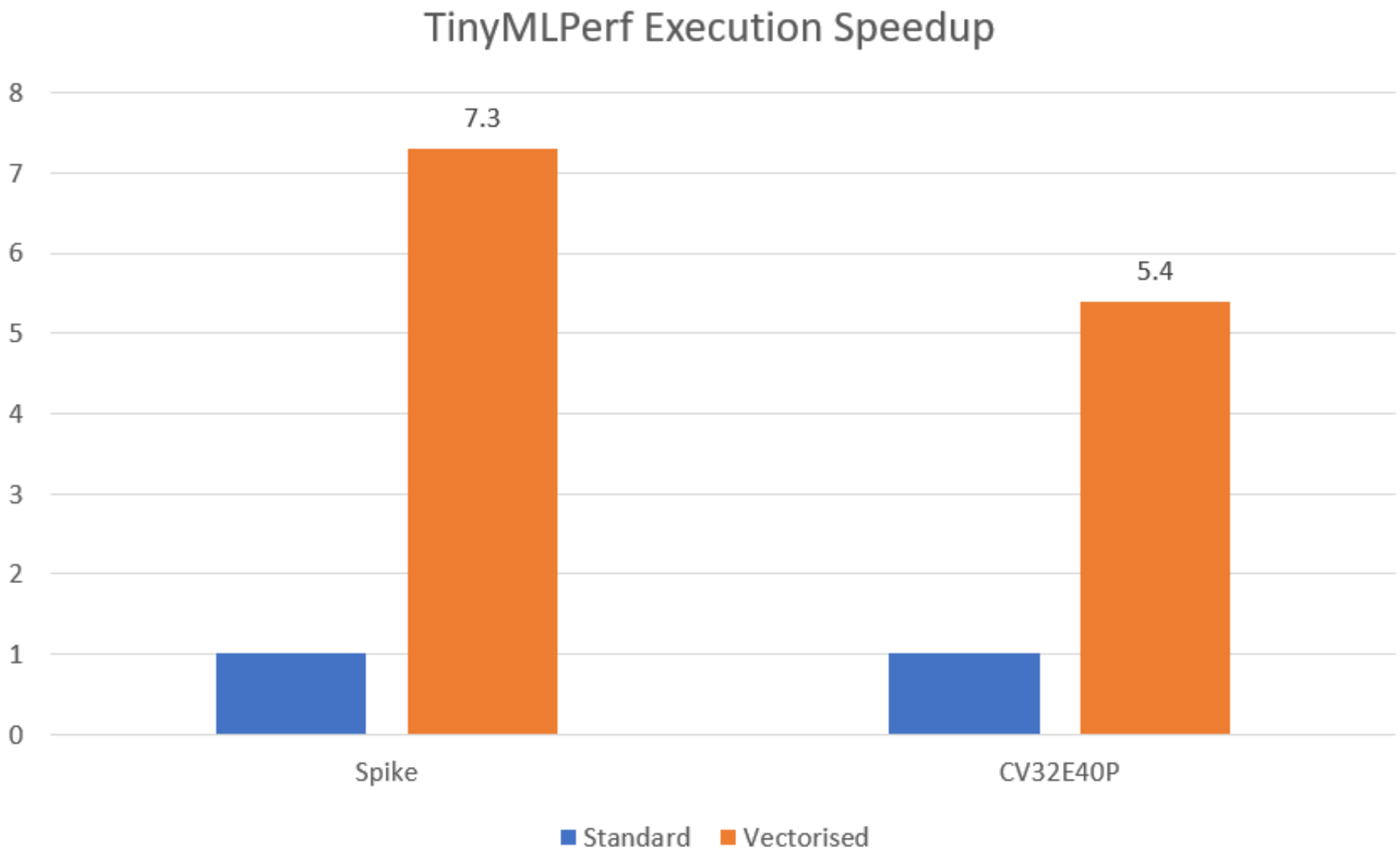
UNIVERSITY OF Southampton

Benchmarking - TFL Micro Optimisation

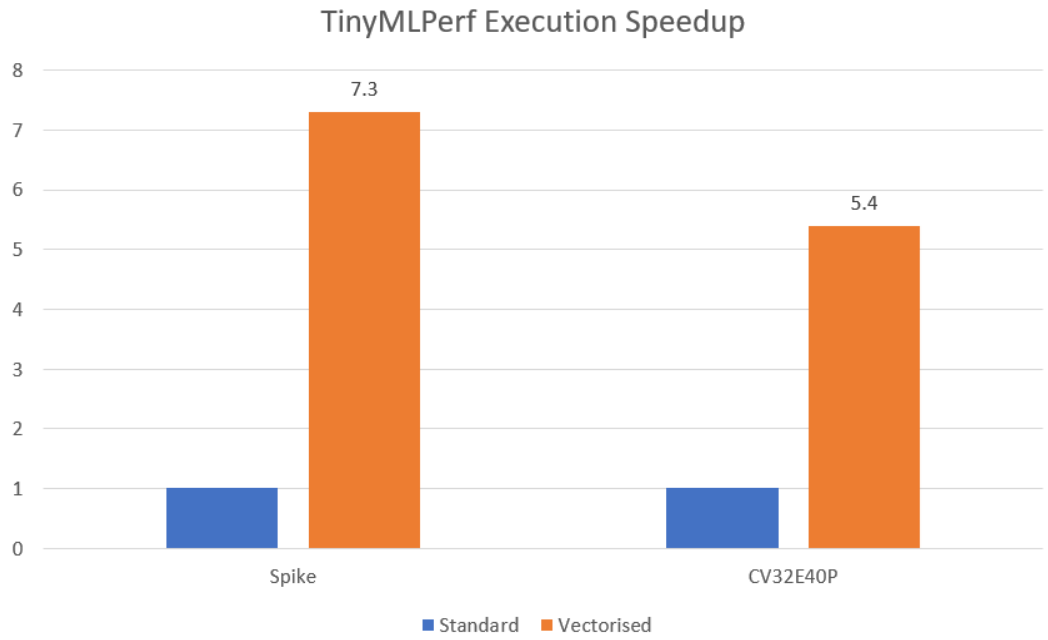
- Workflow
  - Select Operation
  - Edit functions in corresponding CPP file
  - Test correctness of each edit using existing unit tests
  - Test speedup using TinyMLPerf

- Considerations
  - Maintain function interfaces
  - Which operations need optimisation

Benchmarking - Results



Benchmarking - Results



	TinyMLPerf Outputs			
	Invocation 1		Invocation 2	
	Output 1	Output 2	Output 1	Output 2
Expected	0.066406	0.933594	0.781250	0.218750
Acc. CV32	0.058594	0.941406	0.781250	0.218750

UNIVERSITY OF Southampton

Conclusion

- 5.4x performance increase in TinyMLPerf
- Only a few RVV instructions
- Easy to integrate custom instructions into TFL Micro
- Accelerator/CPU integration not so easy

UNIVERSITY OF Southampton

All code (vector algorithms, TFL Micro, TinyMLPerf, accelerator)

https://github.com/AI-Vector-Accelerator