



RISE

RISC-V Software Ecosystem

Optimizing Software for RISC-V

RISC-V Summit Europe June 26, 2024

Nathan Egge negge@google.com

RISC-V Software Ecosystem (RISE)

RISE Mission Statement

Accelerate the development of open source software for RISC-V

RISC-V Software Ecosystem (RISE)



RISE Mission Statement

Accelerate the development of open source software for RISC-V

System Libraries WG Charter

*Focus on **high-leverage initiatives** that unblock upstream projects and bring RISC-V to the same functionality and performance as other ISAs*

RISC-V Software Ecosystem (RISE)



RISE Mission Statement

Accelerate the development of open source software for RISC-V

System Libraries WG Charter

*Focus on **high-leverage initiatives** that unblock upstream projects and bring RISC-V to the same functionality and performance as other ISAs*

- In scope:
 - Open source tooling
 - Best practice guides
 - Case studies
 - Writing software optimizations
 - Performance benchmarking
 - Dependency analysis
- Out of scope:
 - Proprietary / vendor extensions
 - Non-ratified RVI extensions
 - Closed source software
 - Private repos / issue trackers

RISC-V Software Ecosystem (RISE)



RISE Mission Statement

Accelerate the development of open source software for RISC-V

System Libraries WG Charter

*Focus on **high-leverage initiatives** that unblock upstream projects and bring RISC-V to the same functionality and performance as other ISAs*

- In scope:
 - Open source tooling
 - Best practice guides
 - Case studies
 - Writing software optimizations
 - Performance benchmarking
 - Dependency analysis
- Out of scope:
 - Proprietary / vendor extensions
 - Non-ratified RVI extensions
 - Closed source software
 - Private repos / issue trackers

RISE Case Study: Adding RVV 1.0 to dav1d



- dav1d is an AV1 decoder
 - Goals: fastest software decoder, cross-platform, small binary size
 - Achieves this through a *LOT* of handwritten assembly!

Totals grouped by language (dominant language first):

```
asm:          234001 (85.33%)
ansic:        40075 (14.61%)
sh:           161 (0.06%)
```

- Essentially a C orchestrator around specialized DSP functions
 - C ABI conformance matters, assembly functions can do weird things
- Good place to test RISC-V Vector assembly routines
 - Excellent testing framework with built-in performance evaluation

[1] <https://code.videolan.org/videolan/dav1d/>

dav1d Correctness Testing



- Unit tests:
 - Driven by `checkasm` test harness
 - For each CPU configuration (e.g., SSE2, SSSE3, SSE4.1, AVX2, AVX-512)
 - For each DSP function
 - Generate random input data with padding
 - Call C function
 - Call ASM function
 - Compare output data
 - Check padding not overwritten
- Integration tests:
 - Driven by `dav1d` command line decoder
 - Decode 772 test videos and compare md5sum
 - Run previously found configurations from fuzzing bugs

Adding RISC-V Vector Specific Functions



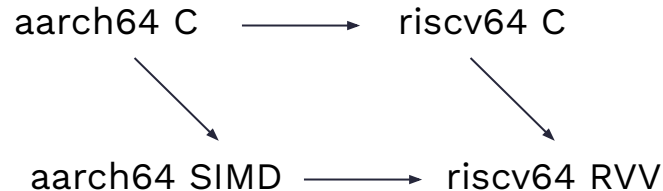
- Implemented a single DSP function for 2D 4x4 transform
 - Used HWCAP for feature detection
 - Hand-written RVV 1.0 assembly, ~200 instructions
- Added C code for new architecture, RISC-V
 - Mostly boiler-plate code, good example of dynamic dispatch
 - Hooks into existing testing infrastructure
- This is enough to test DSP function for correctness

```
nathan@vulpes:/tmp/RISCV-DevEnv-2023-10-27
File Edit View Search Terminal Help
negge@gentoo-riscv ~/git/david/build $ tests/checkasm --function=inv_txfm_add_4x4_identity_identity*8*
checkasm: using random seed 757270960
RVV:
- itx_8bpc.add_4x4          [OK]
checkasm: all 2 tests passed
negge@gentoo-riscv ~/git/david/build $
```



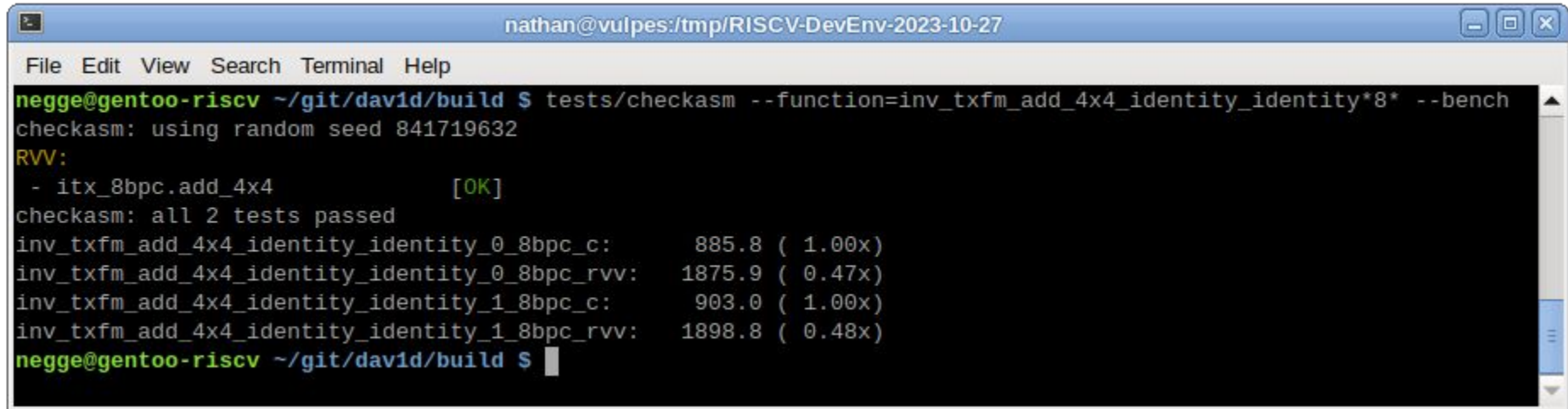

dav1d Performance Testing

- Built-in performance measurement tool
 - Driven by `checkasm` test harness
 - Run C code in a loop N times
 - Run ASM code in a loop N times
 - Compute speed-up
- Can compare RVV hardware e.g., VLEN = 128b vs VLEN = 256b
 - Algorithms that fit in VLEN should get full Nx speedup C -> ASM
- Can compare ISAs by running benchmarks on “equivalent” hardware
 - NEON has 128-bit packed SIMD, expect same speed up C -> ASM for VLEN = 128



dav1d Performance Testing (Cont)

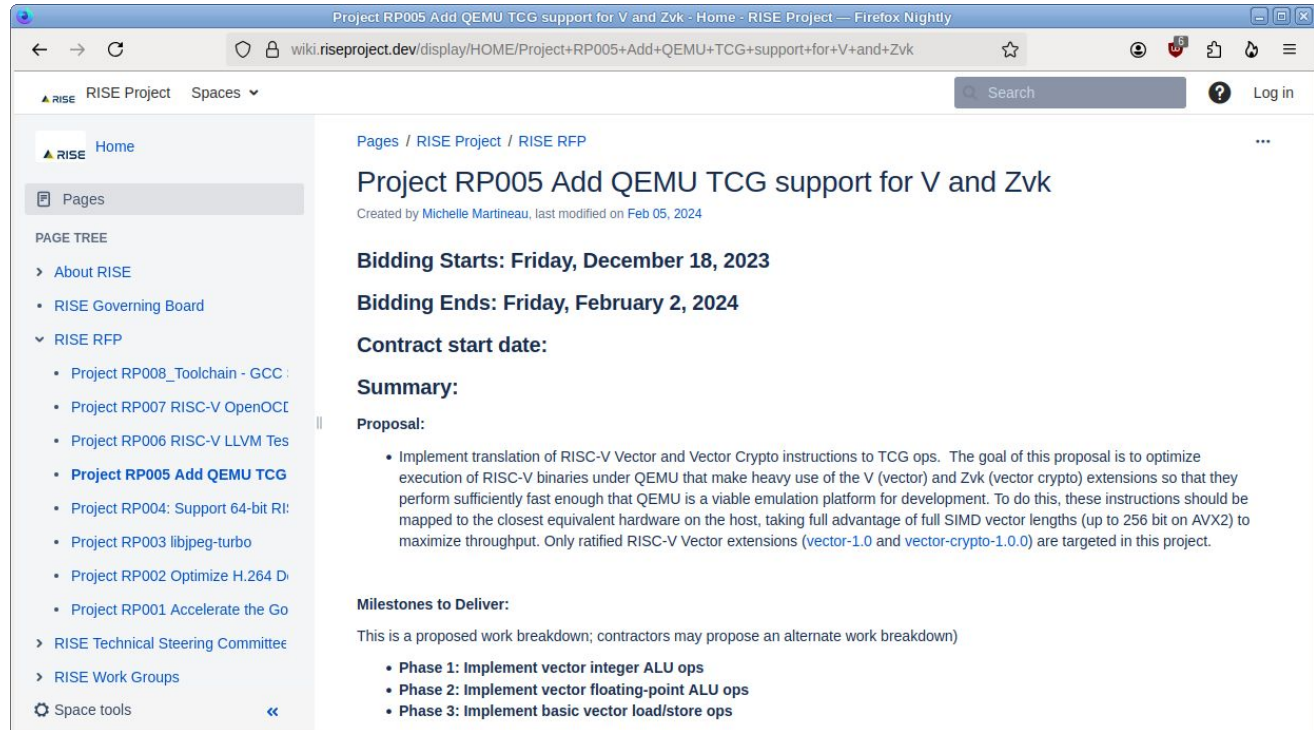
- Easy to run, just add `--bench` to previous `checkasm` command
- Performance benefit was not clear
 - Still had no access to RISC-V Vector 1.0 hardware
 - Under QEMU, **emulated RVV running predictability *slower* than scalar!**



```
nathan@vulpes:/tmp/RISCV-DevEnv-2023-10-27
File Edit View Search Terminal Help
negge@gentoo-riscv ~/git/david/build $ tests/checkasm --function=inv_txfm_add_4x4_identity_identity*8* --bench
checkasm: using random seed 841719632
RVV:
- itx_8bpc.add_4x4 [OK]
checkasm: all 2 tests passed
inv_txfm_add_4x4_identity_identity_0_8bpc_c:      885.8 ( 1.00x)
inv_txfm_add_4x4_identity_identity_0_8bpc_rvv:  1875.9 ( 0.47x)
inv_txfm_add_4x4_identity_identity_1_8bpc_c:      903.0 ( 1.00x)
inv_txfm_add_4x4_identity_identity_1_8bpc_rvv:  1898.8 ( 0.48x)
negge@gentoo-riscv ~/git/david/build $
```

RP005 Add QEMU TCG support for V and Zvk

- Poor QEMU vector performance slows innovation cycle
- RISE recognized ecosystem need and published an RFP



Project RP005 Add QEMU TCG support for V and Zvk - Home - RISE Project — Firefox Nightly

wiki.riseproject.dev/display/HOME/Project+RP005+Add+QEMU+TCG+support+for+V+and+Zvk

RISE RISE Project Spaces

Home

Pages

PAGE TREE

- > About RISE
 - RISE Governing Board
- > RISE RFP
 - Project RP008_Toolchain - GCC
 - Project RP007 RISC-V OpenOCL
 - Project RP006 RISC-V LLVM Tes
 - **Project RP005 Add QEMU TCG**
 - Project RP004: Support 64-bit RI
 - Project RP003 libjpeg-turbo
 - Project RP002 Optimize H.264 D
 - Project RP001 Accelerate the Go
- > RISE Technical Steering Committee
- > RISE Work Groups

Space tools

Pages / RISE Project / RISE RFP

Project RP005 Add QEMU TCG support for V and Zvk

Created by Michelle Martineau, last modified on Feb 05, 2024

Bidding Starts: Friday, December 18, 2023

Bidding Ends: Friday, February 2, 2024

Contract start date:

Summary:

Proposal:

- Implement translation of RISC-V Vector and Vector Crypto instructions to TCG ops. The goal of this proposal is to optimize execution of RISC-V binaries under QEMU that make heavy use of the V (vector) and Zvk (vector crypto) extensions so that they perform sufficiently fast enough that QEMU is a viable emulation platform for development. To do this, these instructions should be mapped to the closest equivalent hardware on the host, taking full advantage of full SIMD vector lengths (up to 256 bit on AVX2) to maximize throughput. Only ratified RISC-V Vector extensions (`vector-1.0` and `vector-crypto-1.0.0`) are targeted in this project.

Milestones to Deliver:

This is a proposed work breakdown; contractors may propose an alternate work breakdown)

- **Phase 1: Implement vector integer ALU ops**
- **Phase 2: Implement vector floating-point ALU ops**
- **Phase 3: Implement basic vector load/store ops**

QEMU Testing Multiple Vector Lengths

- In `.gitlab-ci.yml`

```
test-debian-riscv64:
  extends:
    - .debian-amd64-common
    - .test-common
  needs: [ "build-debian-riscv64" ]
  script:
    - meson setup build --buildtype release
      -Dtestdata_tests=true
      -Dlogging=false
      -Dtrim_dsp=false
      --cross-file package/crossfiles/riscv64-linux.meson
    - ninja -C build
    - cd build && time meson test -v --timeout-multiplier 2
  parallel:
    matrix:
      - QEMU_CPU: [ "rv64,v=true,vext_spec=v1.0,vlen=128,elen=64",
                    "rv64,v=true,vext_spec=v1.0,vlen=256,elen=64",
                    "rv64,v=true,vext_spec=v1.0,vlen=512,elen=64",
                    "rv64,v=true,vext_spec=v1.0,vlen=1024,elen=64" ]
```



QEMU Testing Multiple Vector Lengths (Cont)

test-debian-avx512



test-debian-msan



test-debian-ppc64le



test-debian-riscv64

4

test-debian-riscv64: [rv64,v=true,vext_spec=v1.0,vlen=128,elen=64]



test-debian-tsan



test-debian-riscv64: [rv64,v=true,vext_spec=v1.0,vlen=256,elen=64]



test-debian-ubsan



test-debian-riscv64: [rv64,v=true,vext_spec=v1.0,vlen=512,elen=64]



test-debian-unaligned-stack



test-debian-riscv64: [rv64,v=true,vext_spec=v1.0,vlen=1024,elen=64]



test-win64



Experiment #1

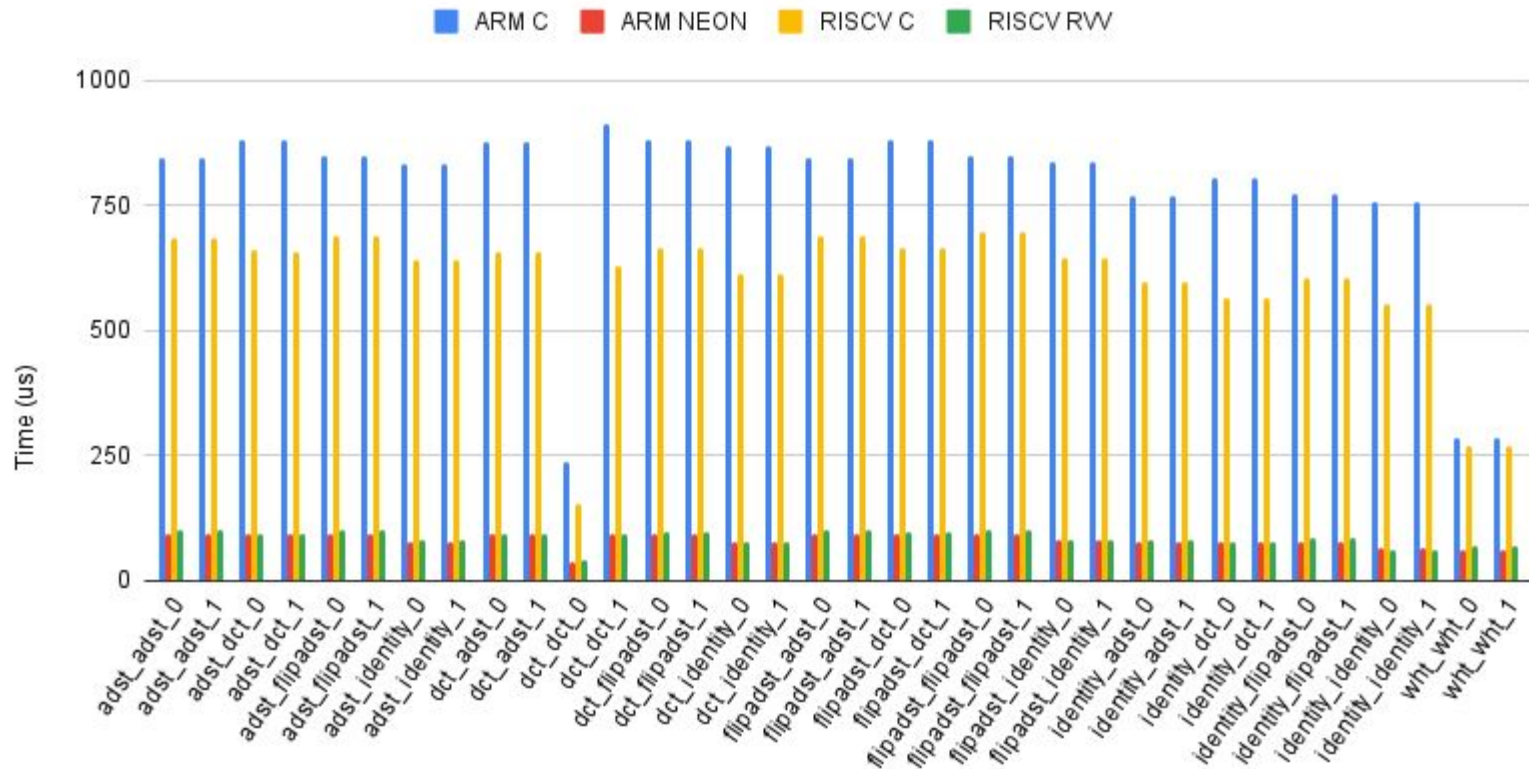
- Run RVV implemented 2D transforms on hardware
 - Kendryte K230
 - Single Core @ 1.6 GHz
 - RVV 1.0 with VLEN = **128 bit**
 - 32k L1 / 256kB L2 / 512MB DDR3
 - ODROID C2
 - Quad Core A53 @ 1.5 GHz
 - Advanced SIMD, aka NEON with **128 bit** registers
 - 32kB L1 / 512kB L2 / 2GB DDR3
- Collect C and ASM timings, compare deltas^{**}

** Warning, not a perfect comparison

- ARM uses `pmccntr_e10` for timings, RISC-V uses `clock_gettime()`
- Differences in CPU frequencies, L2 cache, memory
- Close enough for scalar -> vector verification

4x4 SIMD Comparison (34 functions)

dav1d 4x4 8bpc transforms





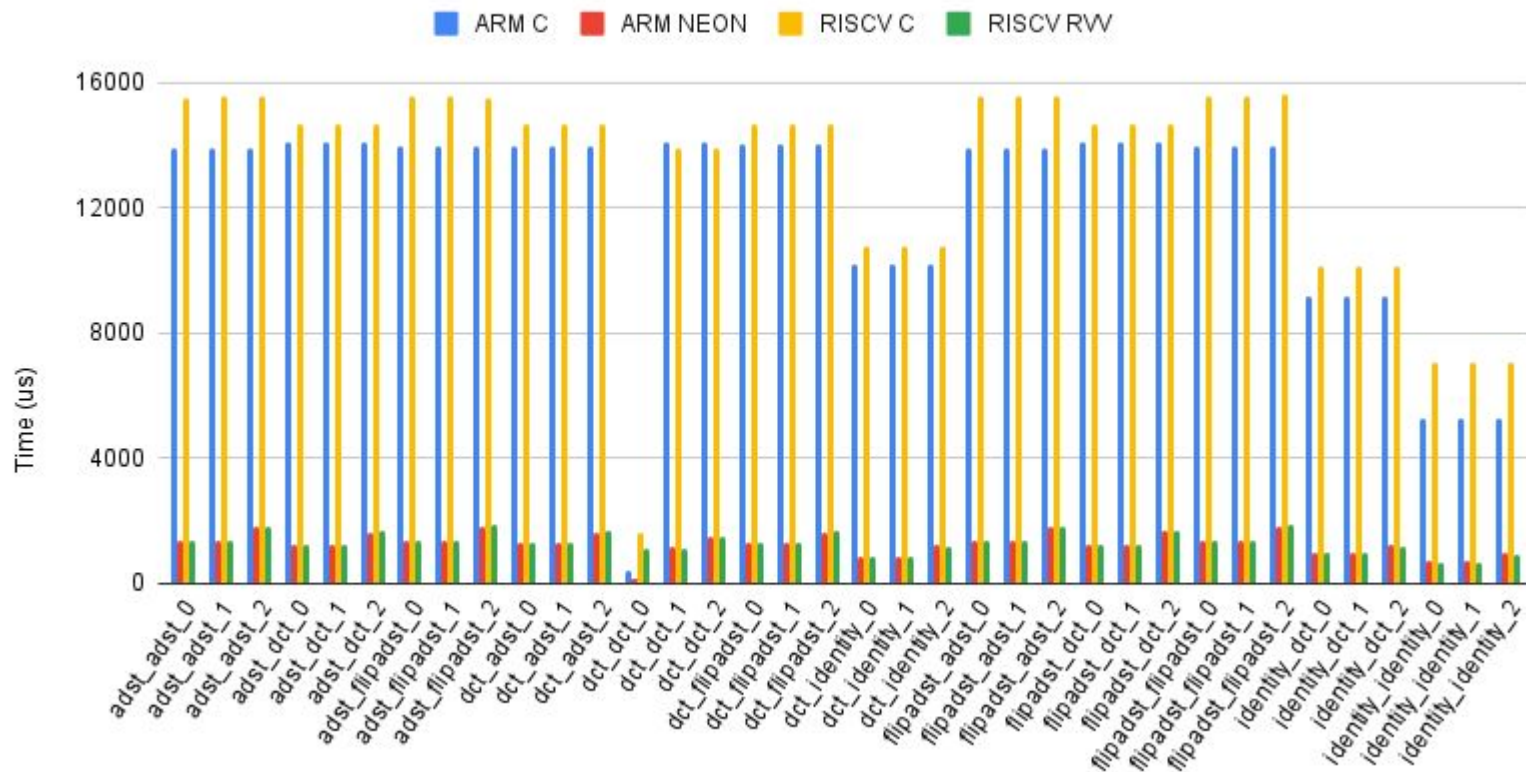
How can this be used?

- Look for functions where speed-up $C \rightarrow ASM$ does not match
- Look for patterns in function complexity that don't match

16x16 SIMD Comparison (36 functions)



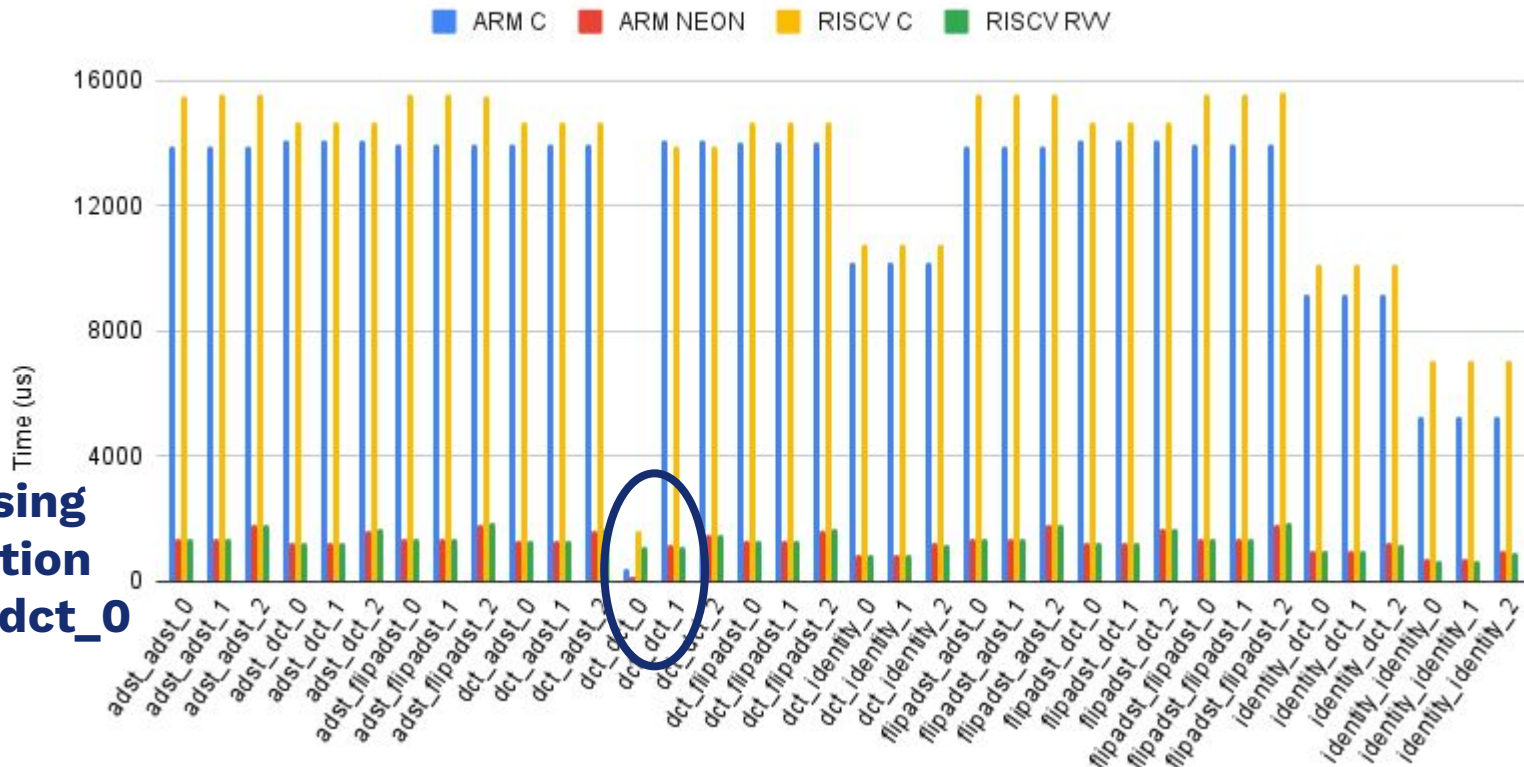
dav1d 16x16 8bpc transforms



16x16 SIMD Comparison (36 functions)



dav1d 16x16 8bpc transforms



**RVV missing optimization
dct_1 -> dct_0**



Experiment #2

- Run motion compensation blend function on hardware
 - Kendryte K230
 - Single Core @ 1.6 GHz
 - RVV 1.0 with VLEN = **128 bit**
 - 32k L1 / 256kB L2 / 512MB DDR3
 - Banana Pi BPI-F3
 - Eight Core @ 1.6 GHz
 - RVV 1.0 with VLEN = **256 bit**
 - 32kB L1 / 512kB L2 / 4GB DDR4
- Collect C and ASM timings, compare deltas^{**}

** Roughly Comparable

- CPUs run at same frequency
- Expect VLEN = 256 to run faster

VLEN 128 v 256 Comparison

- Use checkasm to test blend function

```
void blend(uint8_t *dst, int stride, uint8_t *src, int w, int h, uint8_t *mask);
```

VLEN 128 v 256 Comparison

- Use checkasm to test blend function

```
void blend(uint8_t *dst, int stride, uint8_t *src, int w, int h, uint8_t *mask);
```

**width passed
in a3 register**



VLEN 128 v 256 Comparison

- Use checkasm to test blend function

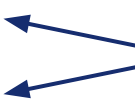
width passed
in a3 register



```
void blend(uint8_t *dst, int stride, uint8_t *src, int w, int h, uint8_t *mask);
```

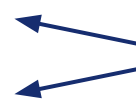
Kendryte K230

blend_w4_8bpc_c:	204.5	(1.00x)
blend_w4_8bpc_rvv:	56.4	(3.62x)
blend_w8_8bpc_c:	608.6	(1.00x)
blend_w8_8bpc_rvv:	87.3	(6.97x)
blend_w16_8bpc_c:	2363.8	(1.00x)
blend_w16_8bpc_rvv:	225.1	(10.50x)
blend_w32_8bpc_c:	5990.3	(1.00x)
blend_w32_8bpc_rvv:	518.8	(11.55x)



Banana Pi BPI-F3

blend_w4_8bpc_c:	201.9	(1.00x)
blend_w4_8bpc_rvv:	59.9	(3.37x)
blend_w8_8bpc_c:	594.9	(1.00x)
blend_w8_8bpc_rvv:	102.4	(5.81x)
blend_w16_8bpc_c:	2308.4	(1.00x)
Blend_w16_8bpc_rvv:	278.3	(8.29x)
blend_w32_8bpc_c:	5854.7	(1.00x)
blend_w32_8bpc_rvv:	569.3	(10.28x)



- Same 1.6GHz freq, same code but VLEN = 256 HW performs **WORSE!**
 - What is going on here?



VLEN 128 v 256 Comparison (Con't)

- Algorithms that use *less* than a full VLEN are essentially tail only
- The `blend_w4` routine operates on blocks 4-wide and N-tall
 - Optimal throughput needs appropriately set LMUL

```
function blend_8bpc_rvv, ext=v
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m2, ta, ma
    j L(blend_epilog)
16: vsetvli zero, a3, e8, m1, ta, ma
    j L(blend_epilog)
8: vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
4: vsetvli zero, a3, e8, mf4, ta, ma
L(blend_epilog):
```





VLEN 128 v 256 Comparison (Con't)

- Algorithms that use *less* than a full VLEN are essentially tail only
- The `blend_w4` routine operates on blocks 4-wide and N-tall
 - Optimal throughput needs appropriately set LMUL

```
function blend_8bpc_rvv, ext=v
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m2, ta, ma
    j L(blend_epilog)
16: vsetvli zero, a3, e8, m1, ta, ma
    j L(blend_epilog)
8: vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
4: vsetvli zero, a3, e8, mf4, ta, ma
L(blend_epilog):
```

```
function blend_vl256_8bpc_rvv, ext=v
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m1, ta, ma
    j L(blend_epilog)
16: vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
8: vsetvli zero, a3, e8, mf4, ta, ma
    j L(blend_epilog)
4: vsetvli zero, a3, e8, mf8, ta, ma
    j L(blend_epilog)
```




VLEN 128 v 256 Comparison (Updated)

- With vector length specific (VLS) implementations specialized on VLEN

Kendryte K230

blend_w4_8bpc_c:	204.5 (1.00x)	
blend_w4_8bpc_rvv:	56.4 (3.62x)	
blend_w8_8bpc_c:	608.6 (1.00x)	
blend_w8_8bpc_rvv:	87.3 (6.97x)	
blend_w16_8bpc_c:	2363.8 (1.00x)	
blend_w16_8bpc_rvv:	225.1 (10.50x)	←
blend_w32_8bpc_c:	5990.3 (1.00x)	
blend_w32_8bpc_rvv:	518.8 (11.55x)	←

Banana Pi BPI-F3

blend_w4_8bpc_c:	201.6 (1.00x)	
blend_w4_8bpc_rvv:	58.0 (3.48x)	
blend_w8_8bpc_c:	595.1 (1.00x)	
blend_w8_8bpc_rvv:	82.1 (7.25x)	
blend_w16_8bpc_c:	2308.8 (1.00x)	
blend_w16_8bpc_rvv:	189.0 (12.22x)	←
blend_w32_8bpc_c:	5853.1 (1.00x)	
blend_w32_8bpc_rvv:	339.5 (17.24x)	←



VLEN 128 v 256 Comparison (Con't)

- Assume Zbb extension present when RVV 1.0 detected
- Rewrite into branchless code using ctz
 - This works because $vsew[5:3] = 0$ when $SEW = e8$

```
function blend_8bpc_rvv, ext=v,zbb
    ctz t0, a3
    addi t0, t0, 0xc4 ←
L(blend_epilog):
    andi t0, t0, 0xc7
    vsetvl zero, a3, t0

    [...]

    ret
endfunc
```

```
function blend_vl256_8bpc_rvv, ext=v,zbb
    ctz t0, a3
    addi t0, t0, 0xc3 ←
    j L(blend_epilog)
endfunc
```



VLEN 128 v 256 Comparison (Con't)

- Assume Zbb extension present when RVV 1.0 detected
- Rewrite into branchless code using ctz
 - This works because $vsew[5:3] = 0$ when SEW = e8

```
function blend_8bpc_rvv, ext=v,zbb
    ctz t0, a3
    addi t0, t0, 0xc4 ←
L(blend_epilog):
    andi t0, t0, 0xc7
    vsetvl zero, a3, t0

    [...]

    ret
endfunc
```

```
function blend_vl256_8bpc_rvv, ext=v,zbb
    ctz t0, a3
    addi t0, t0, 0xc3 ←
    j L(blend_epilog)
endfunc
```

**only 10 bytes
for VLEN = 256
VLS routine!**

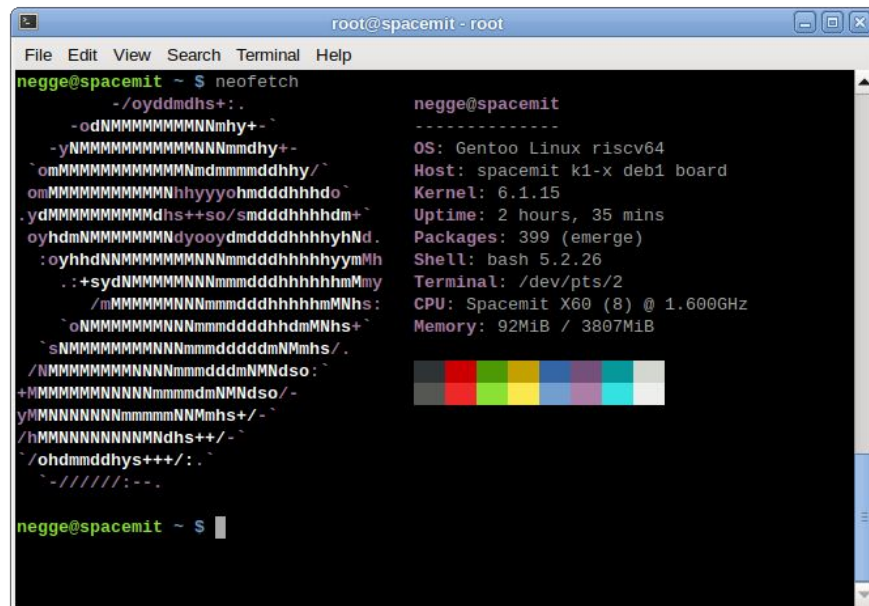
Prebuilt Developer Images

- Facilitate development by providing up-to-date toolchains for building and testing

- Latest toolchain package versions

- clang-18.1.5
- gcc-13.2.1_p20240503
- rust-1.77.1
- binutils-2.42
- cmake-3.29.3
- python-3.12.3
- perl-5.38.2
- git-2.45.1
- Subversion-1.14.3

- Kendryte K230 and Banana Pi BPI-F3



```
root@spacemit - root
File Edit View Search Terminal Help
negge@spacemit ~ $ neofetch
      /oYddmDhs+:/
    -odNMMMMMMMMNNmhy+-
  -yNMMMMMMMMMMMMNNmddhy+-
omMMMMMMMMMMMMMMNdmmdddhhy/
omMMMMMMMMMMMMNNhyyyoHmdddhhdO
.ydMMMMMMMMMdhS++so/smdddhhdhdM+
oyhdmNMMMMMMMMNdYoYdMdddhhdhhyNd.
:oyhdNMMMMMMMMNNmdddhhdhhyMh
.:+sydNMMMMMMMMNNmdddhhdhhdMmY
./mMMMMMMMMNNmdddhhdhhdMNs:
  oNMMMMMMMMNNmdddhhdMNs+
  sNMMMMMMMMNNmdddhhdMNs/
./NMMMMMMMMNNmdddhhdMNsO:
+MMMMMMMMNNmdddhhdMNsO/-
yMMMMMMMMNNmdddhhdMNs+/-
/hMMMMMMMMNNmdddhhdMNs+/-
`ohdmmddhys++:/`
`-/////:--`

negge@spacemit
-----
OS: Gentoo Linux riscv64
Host: spacemit k1-x deb1 board
Kernel: 6.1.15
Uptime: 2 hours, 35 mins
Packages: 399 (emerge)
Shell: bash 5.2.26
Terminal: /dev/pts/2
CPU: Spacemit X60 (8) @ 1.600GHz
Memory: 92MiB / 3807MiB
```

[1] <https://people.videolan.org/~negge/canaan-3G-2024-04-08.img.xz>

[2] <https://people.videolan.org/~negge/spacemit-4G-2024-05-15.img.xz>

**ROMA II image
coming soon!!**



Conclusions

- Benchmarks are a *powerful* tool and **essential** when developing performance optimizations
- Test on multiple VLEN to ensure performance working as expected
- QEMU can and should be used for correctness testing in CI
- Understand the impact of LMUL on throughput (<-- this is critical!) and specialize on VLEN where possible
- Always test performance on *representative* hardware
 - Access to more RVV implementations needed for verification!
- Possible to do ISA <-> ISA and VLEN <-> VLEN comparisons
- Use the latest compilers, toolchains, binutils, etc. when testing

Resources

- RISC-V Optimization Guide
 - <https://gitlab.com/riseproject/riscv-optimization-guide>
- RISE Case Study: Adding RVV 1.0 to dav1d
 - [Part 1 - 2023-Oct-31](#), [Part 2 - 2023-Nov-14](#) and [Part 3 - 2024-Mar-14](#)
 - <https://code.videolan.org/videolan/dav1d/-/tree/master/tests/checkasm> <-- BSD-2
- Prebuilt Gentoo Linux developer images
 - Blog post coming soon <https://riseproject.dev/blog/>
- **Join us!** System Libraries WG meets every other week on Tuesday
 - 8am PDT / 11am EDT / 4pm BST / 5pm CEST / 11pm CST
 - <https://wiki.riseproject.dev/display/HOME/System+Libraries+WG>
- Demos available at the RISE Lounge!

Questions?

