RISC-V optimisations in FFmpeg

History and state of RISC-V Vector for OSS multimedia

Rémi Denis-Courmont

Ixelles, Belgium, 31st January 2026

Outline

1. History
2. How to develop FFmpeg RISC-V optimisations
3. RISC-V Vector pain points

Attendees advisory

Disclaimer

The opinions expressed therein solely represent the personal views of the author.

Attendees advisory

Disclaimer

The opinions expressed therein solely represent the personal views of the author.

- I speak fast.
- I do not articulate well.

Attendees advisory

Disclaimer

The opinions expressed therein solely represent the personal views of the author.

- I speak fast.
- I do not articulate well.

If you did not understand. . .

Do interrupt me if needed!

Who am I?

- FOSDEM
  - 18th time attending (since 2004). . .
  - 2nd scheduled talk

Who am I?

- FOSDEM
    - 18th time attending (since 2004)...
    - 2nd scheduled talk
- Open-source
    - (Mostly) in free time
    - RISC-V maintainer for FFmpeg
    - Core developer of VLC media player

Who am I?

- FOSDEM
  - 18th time attending (since 2004)...
  - 2nd scheduled talk
- Open-source
  - (Mostly) in free time
  - RISC-V maintainer for FFmpeg
  - Core developer of VLC media player
- Daytime: not relevant today.

Outline

1. History
2. How to develop FFmpeg RISC-V optimisations
3. RISC-V Vector pain points

2022

It begins

- Until then: build fixes from Debian & OpenBSD

2022

It begins

- Until then: build fixes from Debian & OpenBSD
- Winter
  - first RISC-V dev board (without V)
  - FFmpeg's FATE automatic testing runs nightly
  - VLC simple vector-optimised loops on SPIKE + pk

2022

It begins

- Until then: build fixes from Debian & OpenBSD
- Winter
  - first RISC-V dev board (without V)
  - FFmpeg's FATE automatic testing runs nightly
  - VLC simple vector-optimised loops on SPIKE + pk
- Summer
  - FFmpeg optimisations start
  - User-mode QEMU supports RVV 1.0

2022

It begins

- Until then: build fixes from Debian & OpenBSD
- Winter
  - first RISC-V dev board (without V)
  - FFmpeg's FATE automatic testing runs nightly
  - VLC simple vector-optimised loops on SPIKE + pk
- Summer
  - FFmpeg optimisations start
  - User-mode QEMU supports RVV 1.0
- Autumn
  - SiFive U74 ⇒ first Zba/Zbb hardware and optimisations
  - `checkasm` test harness implemented

2023

It gets busy

- Spring
  - SiFive patch (not sure if benched on simulator or FPGA)

[1] Institute of Software, Chinese Academy of Science

2023

It gets busy

- Spring
    - SiFive patch (not sure if benched on simulator or FPGA)
- Summer
    - T-Head C910 ⇒ first benchmarks, sort of
        - Macros and kludges to *backport* to RVV 0.7.1 draft spec
        - Some optimisations disabled (e.g. fractional multiplier)

[1] Institute of Software, Chinese Academy of Science

2023

It gets busy

- Spring
  - SiFive patch (not sure if benched on simulator or FPGA)
- Summer
  - T-Head C910 ⇒ first benchmarks, sort of
    - Macros and kludges to *backport* to RVV 0.7.1 draft spec
    - Some optimisations disabled (e.g. fractional multiplier)
- Autumn
  - T-Head C908 ⇒ first *working* hardware
  - Work on optimised fixed-size kernels start
  - ISCAS[1] starts submitting patches

[1] Institute of Software, Chinese Academy of Science

2024

Slow but steady

- Winter
  - RISE funds AVC/H.264 video decoder optimisations

2024

Slow but steady

- Winter
  - RISE funds AVC/H.264 video decoder optimisations
- Spring
  - SpacemiT X60 ⇒ first 256-bit vectors
  - *VL* vs *VLMAX* controversy kicks in

2025

AI takes over and nobody cares anymore

- SiFive X280 ⇒ first 512-bit vectors
  - €€€€ (or rather $$$$)
  - not available for benchmarks

2025

AI takes over and nobody cares anymore

- SiFive X280 ⇒ first 512-bit vectors
    - €€€€ (or rather $$$$)
    - not available for benchmarks
- Development slows down
    - One contributor is reassigned.
    - Another one quits their job.
    - Only one volunteer is left (me).
    - No one likes to do code review (surprise surprise)...

2025

AI takes over and nobody cares anymore

- SiFive X280 ⇒ first 512-bit vectors
  - €€€€ (or rather $$$$)
  - not available for benchmarks
- Development slows down
  - One contributor is reassigned.
  - Another one quits their job.
  - Only one volunteer is left (me).
  - No one likes to do code review (surprise surprise). . .
  - . . . especially not long assembler

Outline

1. History
2. How to develop FFmpeg RISC-V optimisations
3. RISC-V Vector pain points

Compiling FFmpeg

To cross-compile or not to cross-compile

- GCC cross-compiler (or Clang)
- user-mode QEMU with `binfmt_support`

Compiling FFmpeg

To cross-compile or not to cross-compile

- GCC cross-compiler (or Clang)
- user-mode QEMU with `binfmt_support`

In 4 lines

```
git clone \
 https://code.ffmpeg.org/FFmpeg/FFmpeg.git
cd FFmpeg
./configure --enable-cross-compile --arch=riscv \
 --cc="riscv64-linux-gnu-gcc -static" \
 --cxx="riscv64-linux-gnu-g++ -static" \
make fate-checkasm
```

Or you can build natively.

Optimisation work flow

1. Find an unoptimised DSP function.

Optimisation work flow

1. Find an unoptimised DSP function.
2. Refactor the C reference code until it looks like loop.

Optimisation work flow

1. Find an unoptimised DSP function.
2. Refactor the C reference code until it looks like loop.
3. Write assembler equivalent.
4. Recompile.
5. Run `tests/checkasm/checkasm --test=`*foobardsp*

Optimisation work flow

1. Find an unoptimised DSP function.
2. Refactor the C reference code until it looks like loop.
3. Write assembler equivalent.
4. Recompile.
5. Run `tests/checkasm/checkasm --test=`*`foobardsp`*
6. Fail.
7. Debug.
8. Go to Recompile.

Optimisation work flow

1. Find an unoptimised DSP function.
2. Refactor the C reference code until it looks like loop.
3. Write assembler equivalent.
4. Recompile.
5. Run `tests/checkasm/checkasm --test=`*`foobardsp`*
6. Fail.
7. Debug.
8. Go to Recompile.
9. . . .
10. Boast about your benchmarks on ffmpeg-devel@f.o.

What (not) to do

Useful tips

- Do NOT:
  - Use C intrinsics.

What (not) to do

Useful tips

- Do NOT:
  - Use C intrinsics.
  - Use C inline assembler.

What (not) to do

Useful tips

- Do NOT:
  - Use C intrinsics.
  - Use C inline assembler.
  - Optimise functions without unit test cases.

What (not) to do

Useful tips

- Do NOT:
  - Use C intrinsics.
  - Use C inline assembler.
  - Optimise functions without unit test cases.
  - Specialise optimisations by vector length.

What (not) to do

Useful tips

- Do NOT:
    - Use C intrinsics.
    - Use C inline assembler.
    - Optimise functions without unit test cases.
    - Specialise optimisations by vector length.
- Do:
    - Use *Zba* instructions such as `SH1ADD`.
    - Avoid data dependencies between consecutive instructions.
    - Check benchmarks.

Assembler

Assembler good, intrinsics bad

- FFmpeg (+ x264 + dav1d) historically favour assembler
- General suitability problems
  - extraneous register moves or spills
  - CPP[2] headers incompatible with runtime detection
  - assembler used in reference specifications
- RVV special
  - group multiplier (LMUL) ⇐ V-register pressure
  - need to control register allocation

[2] C preprocessor

checkasm

https://code.videolan.org/videolan/checkasm

- Yet another unit test framework

checkasm

https://code.videolan.org/videolan/checkasm

- Yet another unit test framework. . . not really
  - Enumerates all possible (supported) CPU features
  - Randomises inputs
  - Tests optimisations against reference C code
  - Micro-benchmarks
  - Validates ABI conformance
- Originates in x264 project for 586/686 optimisations
- Available separately (BSD 2-clause) from code.VideoLAN.org

Getting involved

FFmpeg or not FFmpeg

- High barrier of entry for remaining work
  - missing test cases,
  - complex cases
  - or both

Getting involved

FFmpeg or not FFmpeg

- High barrier of entry for remaining work
  - missing test cases,
  - complex cases
  - or both
- Non-technical aspects
  - poor reviewer availability
  - strained and difficult community
- Plenty of other computational OSS projects to help!

Outline

1. History
2. How to develop FFmpeg RISC-V optimisations
3. RISC-V Vector pain points
   - Implementation issues
   - Specification gaps

VL vs VLMAX (1/2)

Refresher

Effective multiplier		Vector registers		Maximum element width	
		Count	Size		
`mf8`	$1/8$	32	$VLEN/8$	e8	8 bits
`mf4`	$1/4$		$VLEN/4$	e16	16 bits
`mf2`	$1/2$		$VLEN/2$	e32	32 bits
`m1`	1	32	$VLEN$	e64	64 bits
`m2`	2	16	$VLEN*2$	e64	64 bits
`m4`	4	8	$VLEN*4$		
`m8`	8	4	$VLEN*8$		

$$VLMAX = VLEN * LMUL/SEW$$

$$0 \leq VL \leq VLMAX$$

VL vs VLMAX (2/2)

Round 2

- Intent of the Vector specification:
  - *LMUL* adjusted to register pressure.
  - Execution time scaled to *VL*.
  - Doubling vector length halves execution time.

[3] at constant *VL* and *LMUL*

VL vs VLMAX (2/2)

Round 2

- Intent of the Vector specification:
  - *LMUL* adjusted to register pressure.
  - Execution time scaled to *VL*.
  - Doubling vector length halves execution time.
- Quality-challenged implementations:
  - Execution time scaled to *VLMAX*.
  - Doubling vector length keeps same execution time[3]
- Specialisations for each *VLEN* are *intractable*.
- Already 3 lengths commercially available (128, 256, 512).

[3] at constant *VL* and *LMUL*

Segmented loads & stores

- Strides
  - *Unit-strided* loads & stores hit *a single* address, and
  - *Strided* loads & stores hit *VL* addresses . . .
  - independent of the number of segments (1-8).

Segmented loads & stores

- Strides
  - *Unit-strided* loads & stores hit *a single* address, and
  - *Strided* loads & stores hit *VL* addresses . . .
  - independent of the number of segments (1-8).
- Segments
  - Transfers *N* times the data into *N* vectors
  - Power of two segments frequently needed.
  - Expected similar performance as *N* single segments.

Segmented loads & stores

- Strides
  - *Unit-strided* loads & stores hit *a single* address, and
  - *Strided* loads & stores hit *VL* addresses . . .
  - independent of the number of segments (1-8).
- Segments
  - Transfers *N* times the data into *N* vectors
  - Power of two segments frequently needed.
  - Expected similar performance as *N* single segments.
- Arm cores handle *multi-structured* loads and stores well.

Transpose

- memory→register transposition: `vlseg`N`e`M`.v`
- register→memory transposition: `vsseg`N`e`M`.v`

Transpose

- memory→register transposition: vlseg$N$eM.v
- register→memory transposition: vsseg$N$eM.v
- register→register transposition: 🤷😞
- 2D transforms: $Y = C.round(C.X)^T$ (where C constant matrix)
- fall-back: spill on stack and use segmented loads (or stores)
- *Zvzip* extension *under development*

Mixed signedness narrowing clips

- signed narrowing clip: `vnclip.wi`
- unsigned narrowing clip: `vnclipu.wi`

[4] pixel colour components

Mixed signedness narrowing clips

- signed narrowing clip: `vnclip.wi`
- unsigned narrowing clip: `vnclipu.wi`
- signed-to-unsigned clip: 🤷😞
- extremely common in video encoding/decoding:
  - 16-bit intermediate arithmetic for 8-bit samples[4]
- fall-back:
  - 3-6 instructions
  - `vmax.vx`,
  - `vnclipu.wi`
  - and some vector width changes

[4] pixel colour components

Integer distance

Also known as absolute difference

2 instructions for floats

```
vfsub.vv v16, v0, v8
vfabs.v v16, v16
```

3 instructions for non-overflowing integers

```
vsub.vv v16, v0, v8
vsub.vv v24, v8, v0
vmax.v v16, v16, v24
```

- 4-6 instructions if widening integers to avoid overflow
- more register pressure
- *Zvabd* extension in *stabilisation*

Changing element width

To change the selected element width *SEW*

vsetvli zero, zero, eSEW, m$LMUL$, ...

- Vector length `vl` preserved.
- Vector type `vtype`: SEW, $LMUL$ and flags reset, but. . .
- New type must preserve the $SEW/LMUL$ ratio.

Changing element width

To change the selected element width SEW

```
vsetvli zero, zero, eSEW, mLMUL, ...
```

- Vector length `vl` preserved.
- Vector type `vtype`: SEW, $LMUL$ and flags reset, but…
- New type must preserve the $SEW/LMUL$ ratio.
- Wish: instruction to change SEW and adjust $LMUL$ implicitly.

Further references

- RISC-V Vector extension version 1.0.
- https://code.ffmpeg.org/FFmpeg/FFmpeg
- https://fate.ffmpeg.org/?query=subarch:riscv64%2F%2F
- https://code.videolan.org/videolan/checkasm

Any questions?