Improvements to RISC-V Vector code generation in LLVM

Luke Lau, **Alex Bradbury**

RISC-V Summit Europe 2025

igalia

RVV codegen development

- Basic experimental RVV enablement ✅
- Enablement of RVV codegen by default ✅
- Expansion of additional RVV extension support ✅
- Further tuning of performance of generated code ← **We are here**

Improving RVV code generation

- Objective: faster execution time!
- Might be achieved by:
  - Avoiding vectorisation when it isn't profitable
  - Reducing overhead such as CSR switching
  - Minimising spilling
  - Better exploiting capabilities of RVV
  - .....

Note: this talk gives an overview of recent improvements covering contributions from many companies.

Non-power-of-two vectorization

- Unique to RVV, the vl vector length register can handle vectors of arbitrary (not just power of two) sizes
- New in LLVM 20: Support for non-power-of-2 vector widths in the SLP (Superword Level Parallelism) vectorizer

Non-power-of-two vectorization

```
struct rgb { float r,g,b; };
void brighten(struct rgb *x, float f) {
  x->r *= f;
  x->g *= f;
  x->b *= f;
}
```

`clang -O3 -march=rva23u64`

```
vsetivli zero, 2, e32, mf2, ta, ma
vle32.v  v8, (a0)
flw      fa5, 8(a0)
vfmul.vf v8, v8, fa0
fmul.s   fa5, fa0, fa5
vse32.v  v8, (a0)
fsw      fa5, 8(a0)
```

`clang -O3 -march=rva23u64 -mllvm -slp-vectorize-non-power-of-2`

```
vsetivli   zero, 3, e32, m1, ta, ma
vle32.v    v8, (a0)
vfmul.vf   v8, v8, fa0
vse32.v    v8, (a0)
```


`vl` tail folding

- GCC already performs tail folding
  - => Avoid the need for a separate loop to handle the "tail" in a stripmined loop by using the vl register.
  - LLVM catching up to enable by default
- Allows elimination of **minimum trip count**
  - Previously needed at least VLMAX elements to take vector loop
  - `vl` tail folding can take vector loop for < VLMAX elements
  - Big improvement on some benchmarks e.g. e8 loops on x264!
- `vl` on second to last iteration breaks many assumptions about predication
  - `ceil(AVL / 2) ≤ vl ≤ VLMAX if AVL < (2 * VLMAX)`

vl tail folding

`clang -O3 -march=rva23u64`

```
    vsetvli a2, zero, e32, m2, ta, ma
.vector_body:
    vl2re32.v       v8, (a5)
    vadd.vi v8, v8, 1
    vs2r.v  v8, (a5)
    add     a5, a5, a3
    bne     a5, a4, .vector_body
    beq     a1, a6, .exit
.middle:
    sh2add  a2, a6, a0
    sh2add  a0, a1, a0
.scalar_tail:
    lw      a1, 0(a2)
    addi    a1, a1, 1
    sw      a1, 0(a2)
    addi    a2, a2, 4
    bne     a2, a0, .LBB0_7
```

`NEW clang -O3 -march=rva23u64 -mllvm -force-tail-folding-style=data-with-evl -mllvm -prefer-predicate-over-epilogue=predicate-else-scalar-epilogue`

```
.vector_body:
    sub     a5, a1, a3
    sh2add  a2, a3, a0
    vsetvli a5, a5, e32, m2, ta, ma
    vle32.v v8, (a2)
    sub     a4, a4, a6
    vadd.vi v8, v8, 1
    vse32.v v8, (a2)
    add     a3, a3, a5
    bnez    a4, .vector_body
    ret
```


libcall expansion

- RVV can efficiently perform common operations like memcpy and memcmp
- Expanding inline to
  - Avoid function call overhead.
  - Avoid potentially costly spilling given lack of callee saved vector registers in standard calling convention.
  - Generate specialised version of the code (e.g. alignment, size known).

libcall expansion

```
void *copy(char *a, char *b) {
  return memcpy(a, b, 16);
}
```

`clang -O3 -march=rva23u64`

```
copy:
    vsetivli        zero, 16, e8, m1, ta, ma
    vle8.v  v8, (a1)
    vse8.v  v8, (a0)
    ret
```

```
int equal(char *a, char *b) {
  return memcmp(a, b, 16) == 0;
}
```

NEW `clang -O3 -march=rva23u64`

```
equal:
    vsetivli zero, 16, e8, m1, ta, ma
    vle8.v v8, (a0)
    vle8.v v9, (a1)
    vmsne.vv v8, v8, v9
    vcpop.m a0, v8
    seqz a0, a0
    ret
```


Improving codegen for newer RVV extensions

- Minimal vector FP16 support added in Zvfhmin
- Minimal vector BF16 added in Zvfbfmin with widening mul-add in Zvfbfwma
- Need to teach LLVM to generate code for these extensions.
  - Ensure we widen to f32 when profitable.
  - Cost model adjustment needed.

Loop vectorizer f32 widening

- Better support for
  - FP16 w/ zvfhmin
  - bfloat16 w/ zvfbfmin + zvfbfwma
- Widens to f32: care needed to be taken with register pressure!

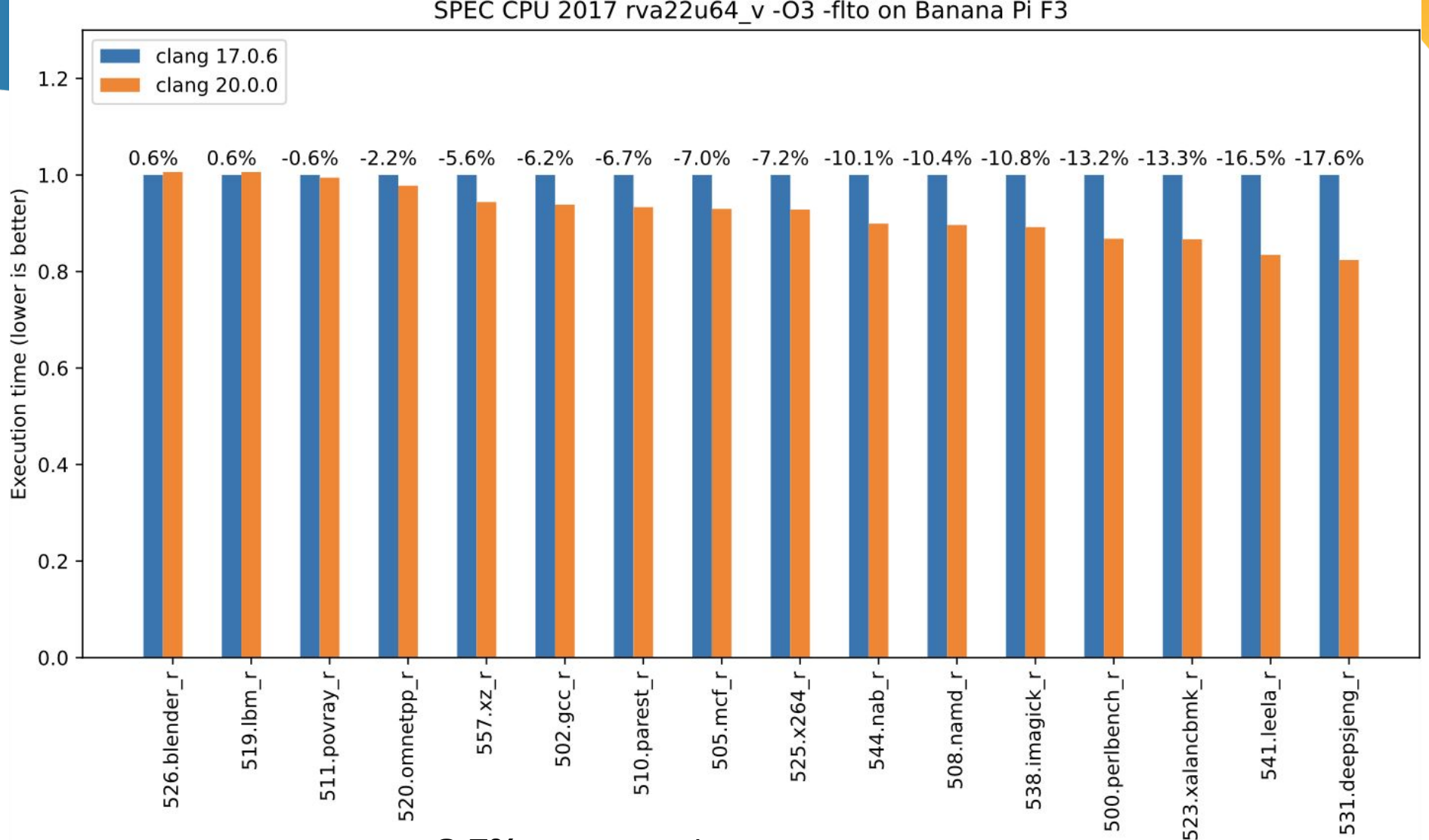
```
void f(float *dst, __bf16 *a, __bf16 *b) {
  for (int i = 0; i < 1024; i++)
    dst[i] += ((float)a[i] * (float)b[i]);
}
```

```
clang -O3 -march=rva23u64
```

```
    vsetvli t4, zero, e16, m1, ta, ma
.LBB0_4:
    vl1re16.v   v8, (t3)
    vl1re16.v   v9, (t2)
    vl2re32.v   v10, (t1)
    vfwmaccbf16.vv  v10, v8, v9
    vs2r.v  v10, (t1)
    add t3, t3, a4
    add t2, t2, a4
    sub t0, t0, a6
    add t1, t1, a7
    bnez    t0, .LBB0_4
```


Other improvements

- `vl` optimizer pass
  - Reduces `vl` to only what is demanded
  - Useful for uarchs that dispatch uops dynamically on `vl`, not LMUL
- RVV support in llvm-exegesis
  - LLVM's tool for automatic benchmarking of microarchitectures
  - Can generate scheduling models for LLVM to use
  - Iterates over all SEW/LMUL/vl/tail policy/mask policy combinations
- `vsetvli` insertion moved to post register allocation
  - `vsetvli` acts as a scheduling barrier everytime `vtype` changes
  - Enables more aggressive scheduling pre register allocation

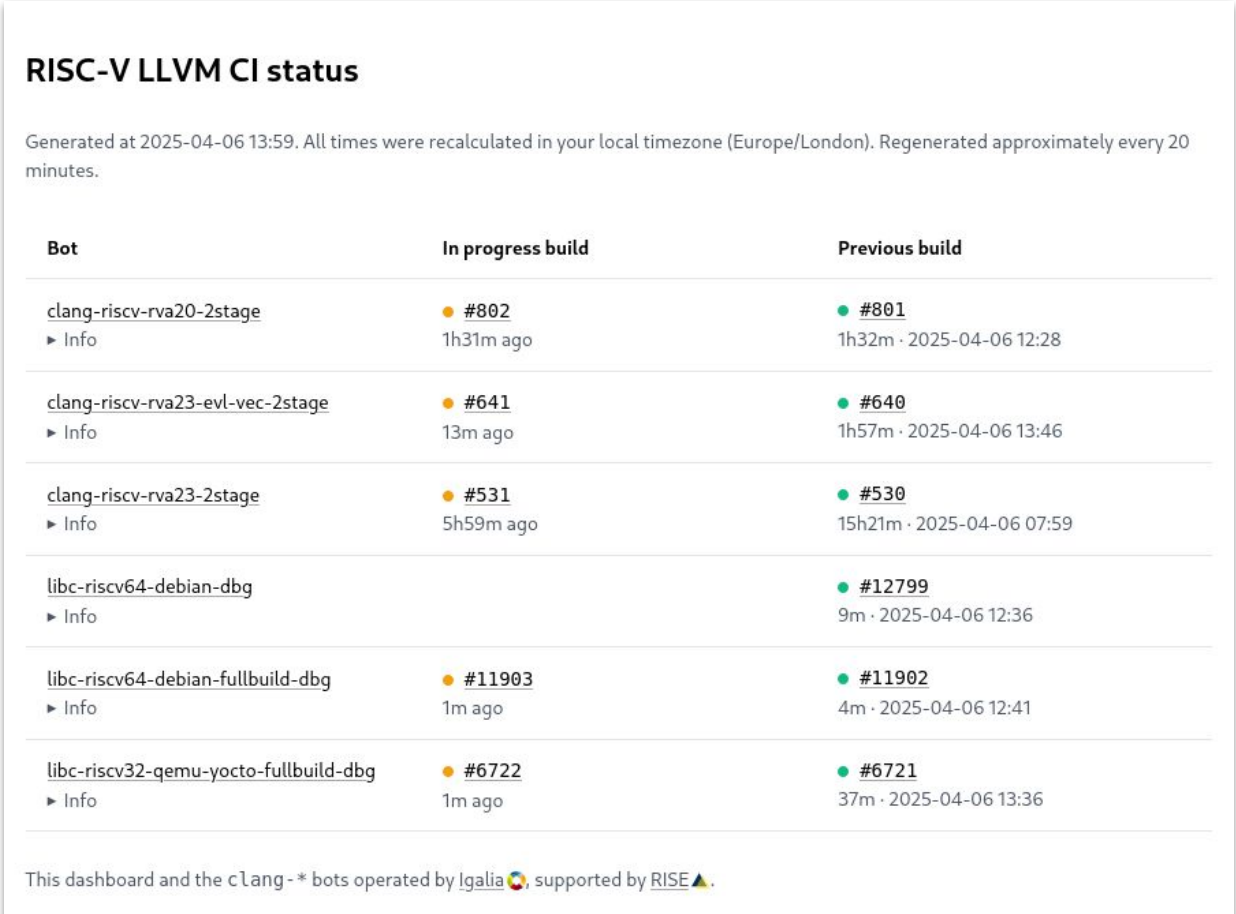


8.7% geomean improvement

Testing

- New CI builders added.
- Targeting new experimental vector configs, e.g. with tail folding.

https://igalia.github.io/riscv-llvm-ci/



RISC-V LLVM CI status

Generated at 2025-04-06 13:59. All times were recalculated in your local timezone (Europe/London). Regenerated approximately every 20 minutes.

Bot	In progress build	Previous build
clang-riscv-rva20-2stage ▸ Info	#802 1h31m ago	#801 1h32m · 2025-04-06 12:28
clang-riscv-rva23-evl-vec-2stage ▸ Info	#641 13m ago	#640 1h57m · 2025-04-06 13:46
clang-riscv-rva23-2stage ▸ Info	#531 5h59m ago	#530 15h21m · 2025-04-06 07:59
libc-riscv64-debian-dbg ▸ Info		#12799 9m · 2025-04-06 12:36
libc-riscv64-debian-fullbuild-dbg ▸ Info	#11903 1m ago	#11902 4m · 2025-04-06 12:41
libc-riscv32-qemu-yocto-fullbuild-dbg ▸ Info	#6722 1m ago	#6721 37m · 2025-04-06 13:36

This dashboard and the `clang-*` bots operated by Igalia, supported by RISE.

Future work

- Smoothing out and enabling more features by default
- Dynamically selecting LMUL in the loop vectorizer
  - Infrastructure to calculate register pressure recently landed
- Early exit vectorization
  - Initial support landed in loop vectorizer via masking
  - Needs support for fault-only-first loads
- Default generic RVV scheduling model
  - In-order cores really need it, especially at higher LMUL
  - Needs to be agreeable: Will affect all of `-march=rva23u64`

Thank you

- Many contributors made this possible
- Andes, ByteDance, Google, Igalia, RISE, Rivos, Samsung, SiFive, SpacemiT, Syntacore, Qualcomm and more!

Questions? asb@igalia.com, luke@igalia.com

Overflow

Challenges

- Performance is highly variable per μarch
  - Some scale with `vl`, others with EMUL, some quadratically e.g. vrgather.vv
  - Exotic memory operations aren't yet optimal in hardware, e.g. strided/indexed
- Register grouping with LMUL introduces a lot of constraints
- Much of underlying infrastructure originally designed with VLS
  - VLA support is beginning to mature thanks to shared infrastructure with AArch64 SVE
- RVV implementations are highly customizable
  - Need to support zvl32b all the way up to zvl65536b
  - True test of RISC-V's unique extensibility!


igalia