OneDNN RISC-V ”V” Primitives

Alexandre Santana, Adrià Armejach, Marc Casas

Barcelona Supercomputing Center
Universitat Politècnica de Catalunya

September 4, 2025

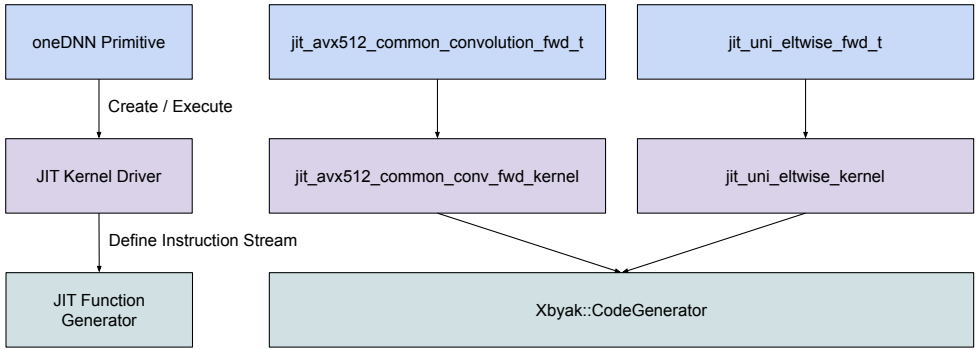
Goals

- Investigate vector-length agnostic algorithms for computer vision workloads;
- Characterize the performance of long- and short-vector ISA implementations;
- Develop the RISC-V "V" (RVV) software ecosystem.

Method

- Translate the x86 AVX512 primitives to RVV;
- Extend the primitives with vector-length agnosticism and register grouping;
- Evaluate and fine-tune for CPU implementations with different vector lengths.

Translate the x86 AVX512 primitives to RVV

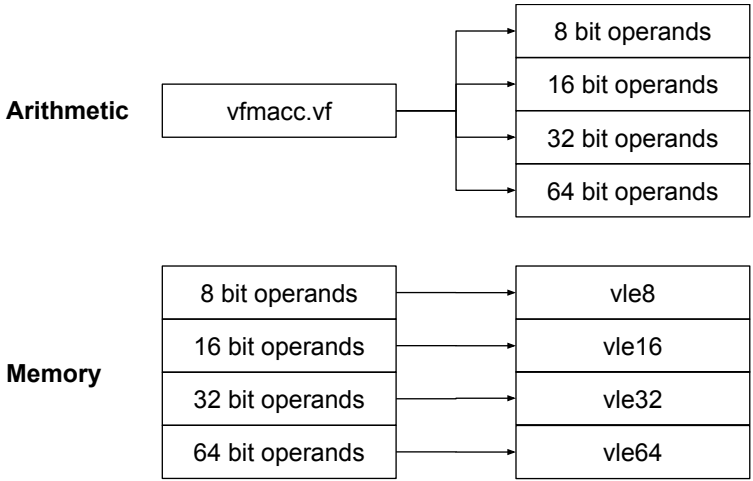


- Develop rvjit, a new JIT assembler library for RISC-V;
- Translate the AVX512 JIT Kernel Drivers to RVV using rvjit;

RISC-V Vector instructions

- Dedicated instruction to configure the vector processing unit traits;
- Application Vector length Request (AVL);
- Application Single-Element-Width (SEW);
- Application Vector Register Grouping (LMUL);
- Granted Vector length Response (GVL);

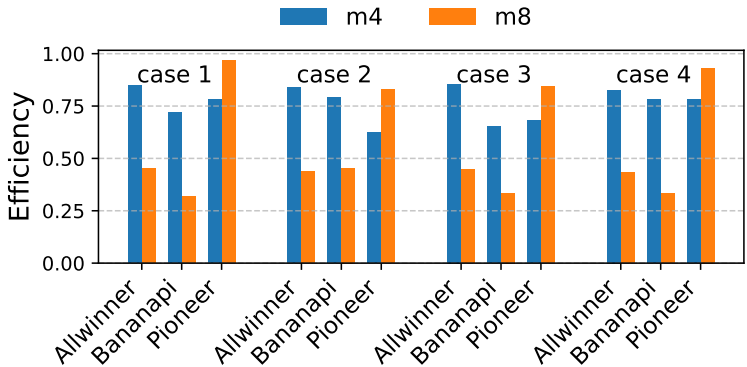
RISC-V Vector instructions



RISC-V Vector instructions

- Vector Register Grouping may be used for performance optimization;
- The RVV standard ISA support group sizes of 1, 2, 4, and 8;
- When LMUL > 1, VLEN is larger, but the ISA exposes fewer vector registers;

LMUL impact on performance

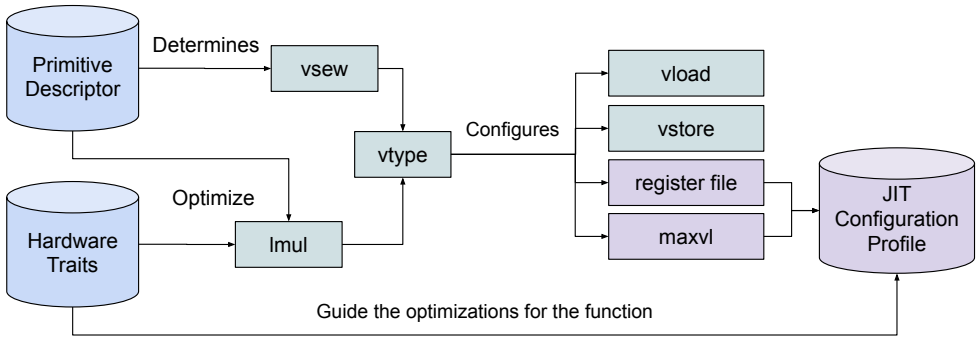


The RVjit library

- Expose mnemonics functions for all rv64gcv instructions;
- Detect extensions and vector traits via cpuinfo and CSRs;
- Implement algorithms that adapt to the vector type traits;

```
const rvjit::registers::gpr gvl = rvjit::registers::x0;
const rvjit::registers::gpr avl = rvjit::registers::a0;
const rvjit::vector::sew vsew   = rvjit::vector::e32;
const rvjit::vector::lmul vlm   = rvjit::vector::m1;
const rvjit::vector::vtype vt = rvjit::vector::vtype(vsew, vlm);
vsetvli(gvl, avl, vt);
```

The RISC-V V JIT driver

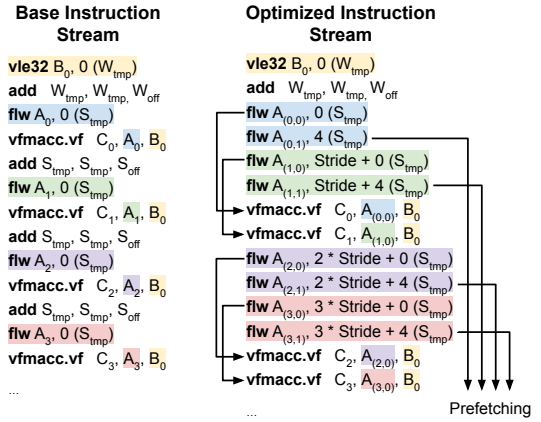


Vector-trait agnostic code generation

```
const auto vsew = rvjit::vector::e32;
const auto vlm = rvjit::vector::m1;
const auto vt = rvjit::vector::vtype(vsew, vlm);
const rvjit::registers::vr* vrf = rvjit::vector::register_file(vlm);
const int nvr = rvjit::vector::register_ngroups(vlm);
vsetvli(gvl, avl, vt);
for (int i = 0; i < nvr-1; ++i) {
    if (i)
        add(dst, dst, off);
    vl(vrf[i], dst, vsew);
}
```

Direct Convolution Code Generation

- State-of-the-art optimizations;
- Vector-length agnostic;
- Optimized vector register grouping;
- Optimized instruction stream for low-end processors;
- Layer-fusion (Elementwise);

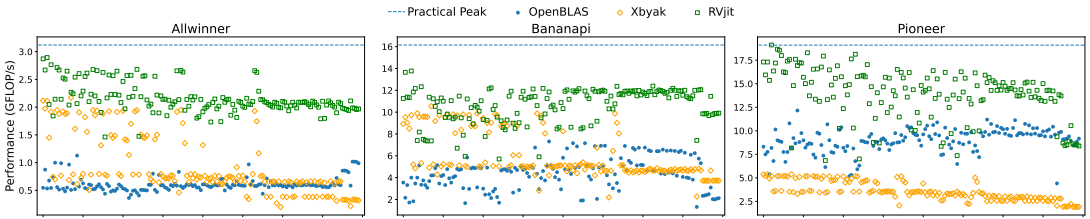


Performance Evaluation Platforms

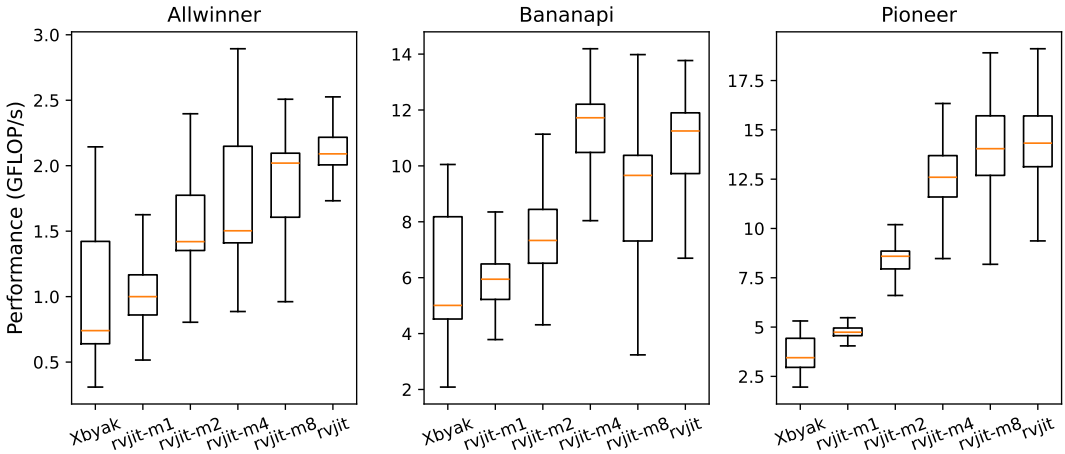
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

System	Allwinner	Bananapi	Pioneer
RV64 Core	C906	SpacemiT K1	C920
Frequency	1.0 GHz	1.6 GHz	2.0 GHz
Performance / core	4 GFLOP/s	25.6 GFLOP/s	32 GFLOP/s
Practical GEMM Peak	3.1 GFLOP/s	16.1 GFLOP/s	18.7 GFLOP/s
Vector Length	128	256	128
RVV version	v0p7_xthead	1.0	v0p7_xthead
L1D cache	32 KB	32 KB	64 KB
L2 / core	-	256 KB	256 KB

Direct Convolution Performance Evaluation

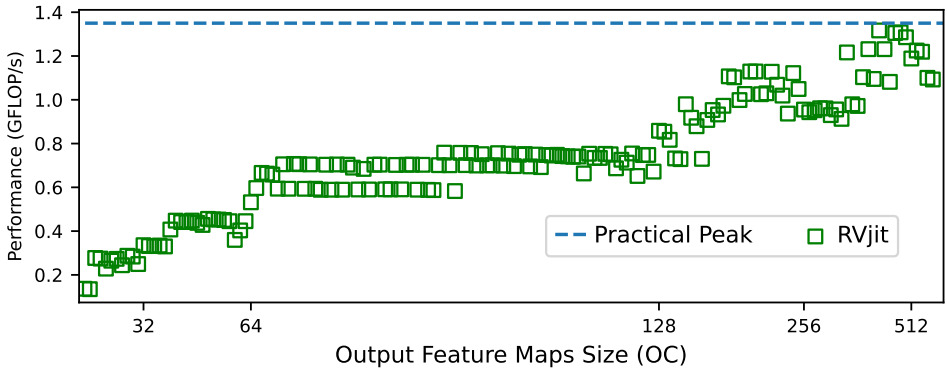


Profiling vector register group sizes



Vector-length agnosticism proof-of-concep

We evaluated our convolution primitive on an FPGA implementing RVV with 16,384-bit vector registers (512 fp32 elements). Our algorithm achieves close to peak performance on problems where $OC \geq VLEN/SEW$.



Takeaways

- We developed and integrated a new RVV JIT assembler library, rvjit, to produce vector-lenth agnostic micro-kernels applicable to different data types and vector grouping settings;
- We implemented high-performance conv and elementwise RVV primitives;
- We validated our primitives for correctness and carried out performance evaluations on CPUs with different vector lengths (128, 256, 16384) and ISA implementations (rvv 0.7.1, rvv 1.0, t-head vector);

Future Work

- Complete convolution API (multi-threading, sum post-op, grouped);
- Contribute to upstream;
- Develop other primitives of interest (Reshape, BNorm, Pool, GEMM);

Thank you!

alexandre.delimassantana@bsc.es