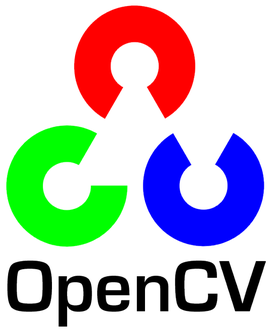


智能软件研究中心
Intelligent Software Research Center

Optimize OpenCV for RISC-V

Google Summer of Code

Mentor: Alexander Smorkalov, Vadim Pisarevsky

Zhang Yin **2020.12.04**

zhangyin2018@iscas.ac.cn

Outline

- Introduction
- Implementation
- Build and Test
- Future Work

Introduction

Project Overview

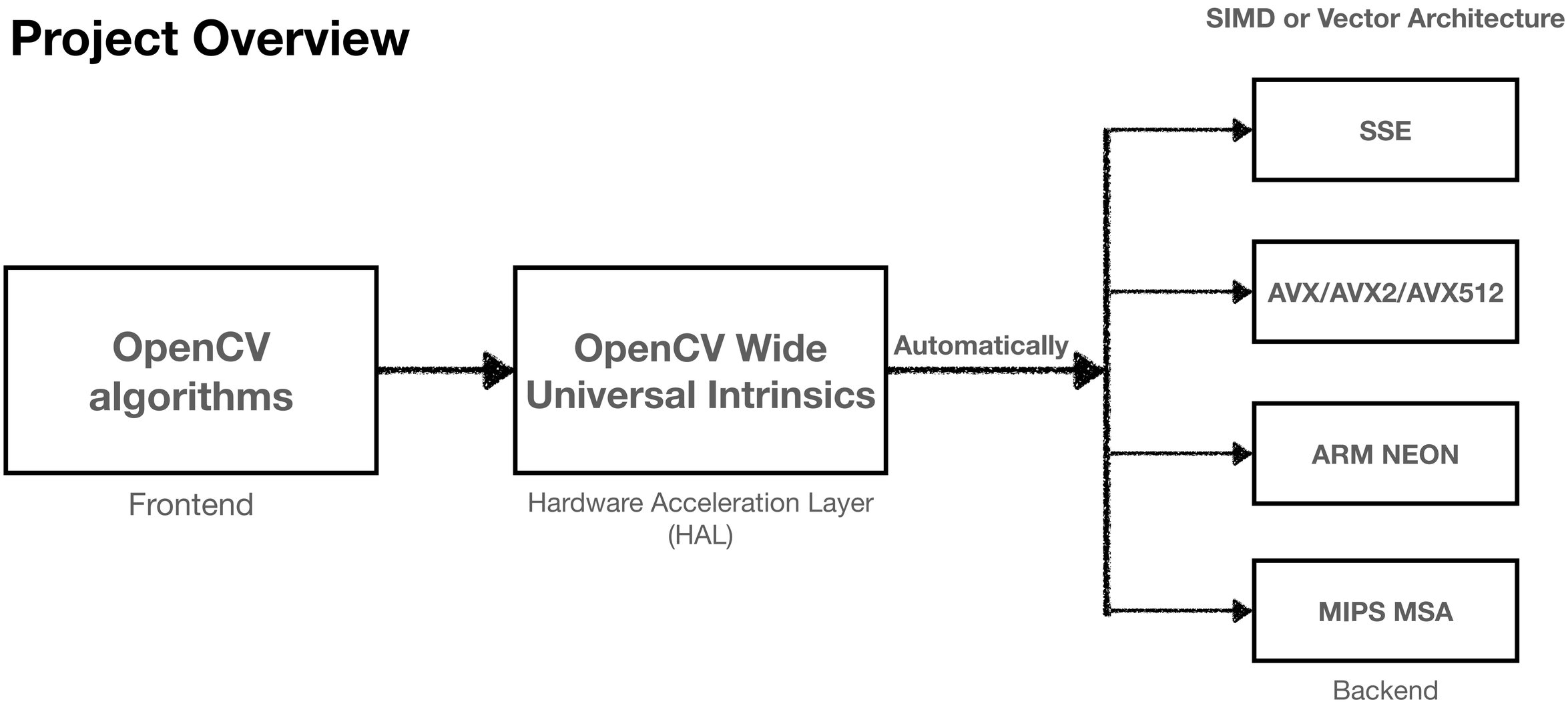
- OpenCV provides a convenient method to port many optimized kernels at once to a new CPU, as long as that CPU supports SIMD/vector instructions. We use so-called Wide Universal Intrinsics for that. By adding implementation of the wide universal intrinsics for RISC-V we can make OpenCV run pretty efficiently on RISC-V architectures.

- Pull request: https://github.com/opencv/opencv/pull/18228
  This pull request has been merged into OpenCV master branch as milestone for OpenCV-4.5.1.

- Guide blogs:
  - ENG: https://plctlab.github.io/opencv/Optimize_OpenCV_for_RISC-V.html
  - CHN: https://zhuanlan.zhihu.com/p/291207654

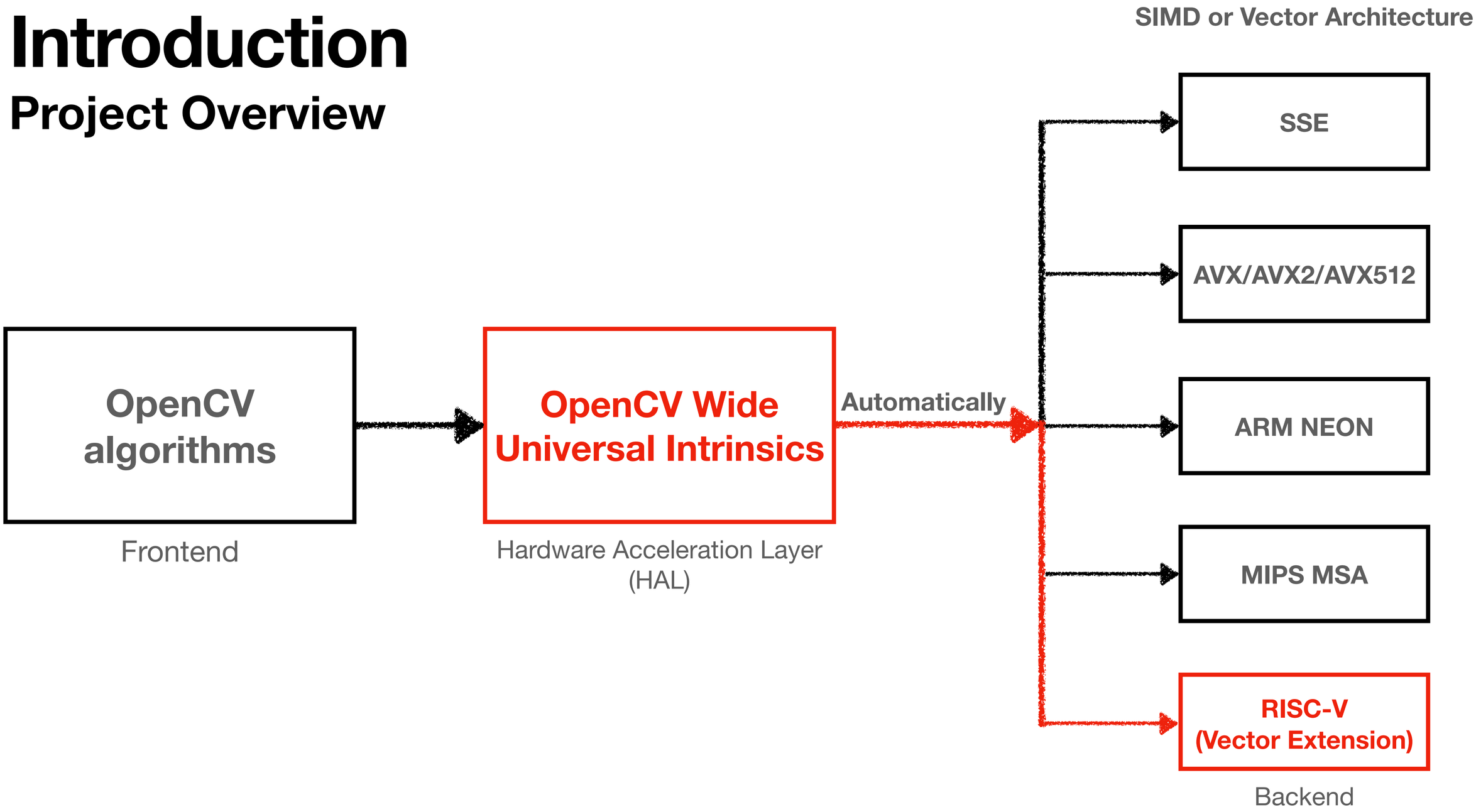
Introduction

Project Overview



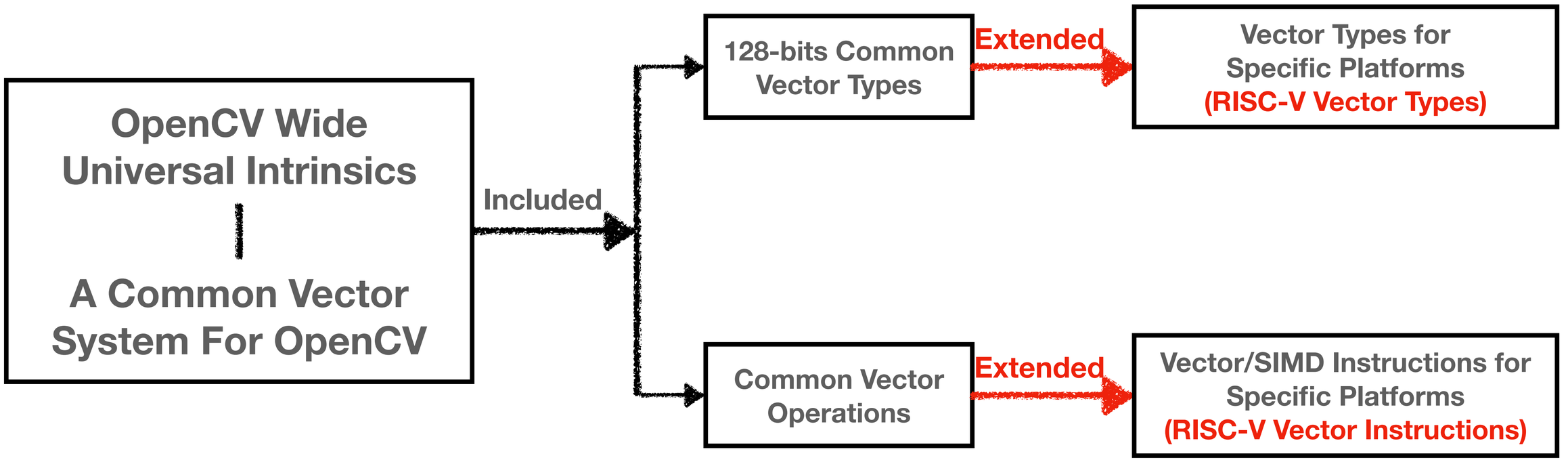
Introduction

Project Overview



Introduction

OpenCV Wide Universal Intrinsics



Introduction
RISC-V ISA

- RISC-V is an open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles.

- Unlike most other ISA designs, the RISC-V ISA is provided under open source licenses that do not require fees to use.

- A number of companies are offering or have announced RISC-V hardware, open source operating systems with RISC-V support are available and the instruction set is supported in several popular software toolchains.

Introduction

RISC-V ISA

Base	Version	Status
RVWMO	2.0	**Ratified**
RV32I	**2.1**	**Ratified**
RV64I	**2.1**	**Ratified**
RV32E	*1.9*	*Draft*
RV128I	*1.7*	*Draft*
Extension	**Version**	**Status**
M	**2.0**	**Ratified**
A	**2.1**	**Ratified**
F	**2.2**	**Ratified**
D	**2.2**	**Ratified**
Q	**2.2**	**Ratified**
C	**2.0**	**Ratified**
Counters	*2.0*	*Draft*
L	*0.0*	*Draft*
B	*0.0*	*Draft*
J	*0.0*	*Draft*
T	*0.0*	*Draft*
P	*0.2*	*Draft*
V	*0.7*	*Draft*
Zicsr	**2.0**	**Ratified**
Zifencei	**2.0**	**Ratified**
Zam	*0.1*	*Draft*
Ztso	*0.1*	*Frozen*

Base modules

Extension modules

RISC-V “V” Vector Extension

- **RISC-V is modular**

- A specific RISC-V ISA contains only one base module and multiple extension modules.
- For example: **RV64IMAFDCV**

Introduction

RISC-V “V” Vector Extension (RVV)

- Being added as a standard extension to the RISC-V ISA
- Still a work in progress, so details might change before standardization
- https://github.com/riscv/riscv-v-spec
- Features:
  - Scalable
  - Dynamic vector types
  - Various vector operations

Introduction

RVV Native Intrinsics

- Intrinsic generally refers to the interface of low-level assembly language in high-level programming language.
- EPI, Sipearl and Sifive published a specification for RISC-V "V" (vector) extension intrinsics, which has been adopted as a standard specification by RISC-V Foundation.
- The goal of this Intrinsic API is to make all instructions of RISC-V "V" (vector) extension accessible to C/C++.
- https://github.com/riscv/rvv-intrinsic-doc

Introduction

Compiler Support — GCC

riscv/riscv-gnu-toolchain

- Branch [rvv-intrinsic] supported by SiFive.

- Current status:
  - Implement ~95% RVV intrinsic function listed in the intrinsic spec (https://github.com/riscv/rvv-intrinsic-doc)
  - FP16 supported for both vector and scalar.
    - fp16 uses __fp16 temporally, this might change in future.
  - Fractional LMUL is not implemented yet.
  - RV32 is not well supported for scalar-vector operations with SEW=64.
  - Function call with vector type is not well supported yet, arguments will be passed/returned in memory in current implementation.
  - *NO* auto vectorization support.

- https://github.com/riscv/riscv-gnu-toolchain

Introduction

Compiler Support — Clang-LLVM

rvv-llvm supported by PLCT Lab

- Supported target RISC-V for 64-bits based on RISC-V “V” extension specification v0.9.

- Current status:
  - 10822 interfaces of RVV intrinsics are provided.
  - All assembly instructions are supported.
  - FP16 is not supported.
  - *NO* auto vectorization support.

- https://github.com/isrc-cas/rvv-llvm

Implementation

Cross-Compilation Environment

- Added RISC-V(RVV) Backend information in following files:

  ```
  cmake/OpenCVCompilerOptimizations.cmake
  modules/core/include/opencv2/core/cv_cpu_dispatch.h
  modules/core/include/opencv2/core/cv_cpu_helper.h
  modules/core/include/opencv2/core/cvdef.h
  modules/core/src/system.cpp
  ```

- Added a simple test for RVV instruction checks:

  ```
  cmake/checks/cpu_rvv.cpp
  ```

- Added toolchain files for target RISC-V with GCC/Clang:

  ```
  platforms/linux/riscv64-gcc.toolchain.cmake
  platforms/linux/riscv64-clang.toolchain.cmake
  ```

Implementation

Cross-Compilation Environment

- `riscv64-gcc.toolchain.cmake`

```
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR riscv64)

set(RISCV_GCC_INSTALL_ROOT /opt/RISCV CACHE PATH "Path to GCC for RISC-V cross compiler installation directory")
set(CMAKE_SYSROOT ${RISCV_GCC_INSTALL_ROOT}/sysroot CACHE PATH "RISC-V sysroot")

set(CMAKE_C_COMPILER ${RISCV_GCC_INSTALL_ROOT}/bin/riscv64-unknown-linux-gnu-gcc)
set(CMAKE_CXX_COMPILER ${RISCV_GCC_INSTALL_ROOT}/bin/riscv64-unknown-linux-gnu-g++)

# Don't run the linker on compiler check
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)

set(CMAKE_C_FLAGS "-march=rv64gcv_zvqmac ${CMAKE_C_FLAGS}")
set(CMAKE_CXX_FLAGS "-march=rv64gcv_zvqmac ${CXX_FLAGS}")

set(CMAKE_FIND_ROOT_PATH ${CMAKE_SYSROOT})
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
```

Implementation

Cross-Compilation Environment

- `riscv64-clang.toolchain.cmake`

```
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR riscv64)

set(RISCV_CLANG_BUILD_ROOT /opt/rvv-llvm CACHE PATH "Path to CLANG for RISC-V cross compiler build directory")
set(RISCV_GCC_INSTALL_ROOT /opt/RISCV CACHE PATH "Path to GCC for RISC-V cross compiler installation directory")
set(CMAKE_SYSROOT ${RISCV_GCC_INSTALL_ROOT}/sysroot CACHE PATH "RISC-V sysroot")

set(CLANG_TARGET_TRIPLE riscv64-unknown-linux-gnu)

set(CMAKE_C_COMPILER ${RISCV_CLANG_BUILD_ROOT}/bin/clang)
set(CMAKE_C_COMPILER_TARGET ${CLANG_TARGET_TRIPLE})
set(CMAKE_CXX_COMPILER ${RISCV_CLANG_BUILD_ROOT}/bin/clang++)
set(CMAKE_CXX_COMPILER_TARGET ${CLANG_TARGET_TRIPLE})
set(CMAKE_ASM_COMPILER ${RISCV_CLANG_BUILD_ROOT}/bin/clang)
set(CMAKE_ASM_COMPILER_TARGET ${CLANG_TARGET_TRIPLE})

# Don't run the linker on compiler check
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)

set(CMAKE_C_FLAGS "-march=rv64gcv0p9 -menable-experimental-extensions --gcc-toolchain=${RISCV_GCC_INSTALL_ROOT} -w
${CMAKE_C_FLAGS}")
set(CMAKE_CXX_FLAGS "-march=rv64gcv0p9 -menable-experimental-extensions --gcc-toolchain=${RISCV_GCC_INSTALL_ROOT} -w ${CXX_FLAGS}")

set(CMAKE_FIND_ROOT_PATH ${CMAKE_SYSROOT})
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
```

Implementation

Method

Implementation

Vector Types

Universal Intrinsics (Fixed 128 bits)		Native Intrinsics (VLEN = 128)
v_uint8x16	→	vuint8m1_t
v_int8x16	→	vint8m1_t
v_uint16x8	→	vuint16m1_t
v_int16x8	→	vint16m1_t
v_uint32x4	→	vuint32m1_t
v_int32x4	→	vint32m1_t
v_uint64x2	→	vuint64m1_t
v_int64x2	→	vint64m1_t
v_float32x4	→	vfloat32m1_t
v_float64x2	→	vfloat64m1_t

Implementation

Vector Operations

Most operations are implemented only for some subset of the available types, following matrices shows the applicability of different operations to the types.

Regular integers:

Operations\Types	uint 8x16	int 8x16	uint 16x8	int 16x8	uint 32x4	int 32x4
load, store	x	x	x	x	x	x
interleave	x	x	x	x	x	x
expand	x	x	x	x	x	x
expand_low	x	x	x	x	x	x
expand_high	x	x	x	x	x	x
expand_q	x	x				
add, sub	x	x	x	x	x	x
add*wrap, sub*wrap	x	x	x	x		
mul_wrap	x	x	x	x		
mul	x	x	x	x	x	x
mul_expand	x	x	x	x	x	
compare	x	x	x	x	x	x
shift			x	x	x	x
dotprod				x		x
dotprod_fast				x		x
dotprod_expand	x	x	x	x		x
dotprod*expand*fast	x	x	x	x		x
logical	x	x	x	x	x	x
min, max	x	x	x	x	x	x
absdiff	x	x	x	x	x	x
absdiffs		x		x		
reduce	x	x	x	x	x	x
mask	x	x	x	x	x	x
pack	x	x	x	x	x	x
pack_u	x		x			
pack_b	x					
unpack	x	x	x	x	x	x
extract	x	x	x	x	x	x
rotate (lanes)	x	x	x	x	x	x
cvt_flt32						x
cvt_flt64						x
transpose4x4					x	x
reverse	x	x	x	x	x	x
extract_n	x	x	x	x	x	x
broadcast_element					x	x

Implementation

Vector Operations

Most operations are implemented only for some subset of the available types, following matrices shows the applicability of different operations to the types.

Big integers:

Operations\Types	uint 64x2	int 64x2
load, store	x	x
add, sub	x	x
shift	x	x
logical	x	x
reverse	x	x
extract	x	x
rotate (lanes)	x	x
cvt_flt64		x
extract_n	x	x

Floating point:

Operations\Types	float 32x4	float 64x2
load, store	x	x
interleave	x	
add, sub	x	x
mul	x	x
div	x	x
compare	x	x
min, max	x	x
absdiff	x	x
reduce	x	
mask	x	x
unpack	x	x
cvt_flt32		x
cvt_flt64	x	
sqrt, abs	x	x
float math	x	x
transpose4x4	x	
extract	x	x
rotate (lanes)	x	x
reverse	x	x
extract_n	x	x
broadcast_element	x	

Implementation

Vector Operations

Load and store operations

These operations allow to set contents of the register explicitly or by loading it from some memory block and to save contents of the register to memory block.

- Constructors:
  @ref v_reg::v_reg(const _Tp *ptr) "from memory",
  @ref v_reg::v_reg(_Tp s0, _Tp s1) "from two values", …

- Other create methods:
  @ref v_setall_s8, @ref v_setall_u8, ...,
  @ref v_setzero_u8, @ref v_setzero_s8, …

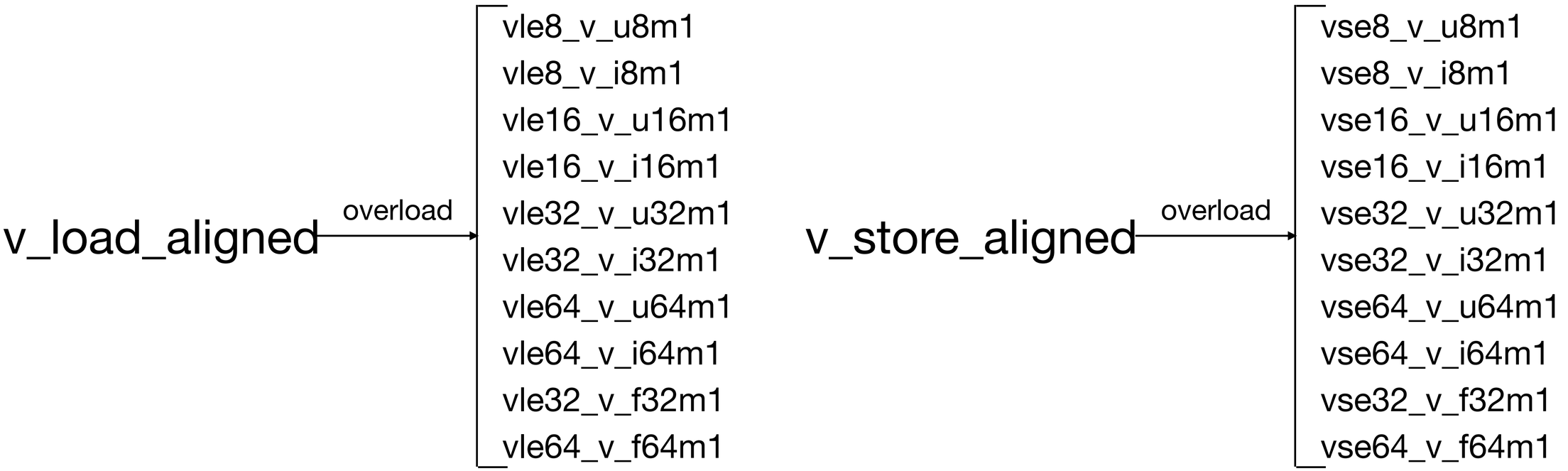
- Memory operations:
  @ref v_load, @ref v_load_aligned, @ref v_load_low, @ref v_load_halves,
  @ref v_store, @ref v_store_aligned,@ref v_store_high, @ref v_store_low

Implementation

Vector Operations

Load and store operations

These operations allow to set contents of the register explicitly or by loading it from some memory block and to save contents of the register to memory block.



Implementation

Vector Operations

Arithmetic, bitwise and comparison operations

Element-wise binary and unary operations.

- Arithmetics:
  @ref operator +(const v_reg &a, const v_reg &b) "+",
  @ref operator -(const v_reg &a, const v_reg &b) "-",
  @ref operator *(const v_reg &a, const v_reg &b) "*",
  @ref operator /(const v_reg &a, const v_reg &b) "/",
  @ref v_mul_expand

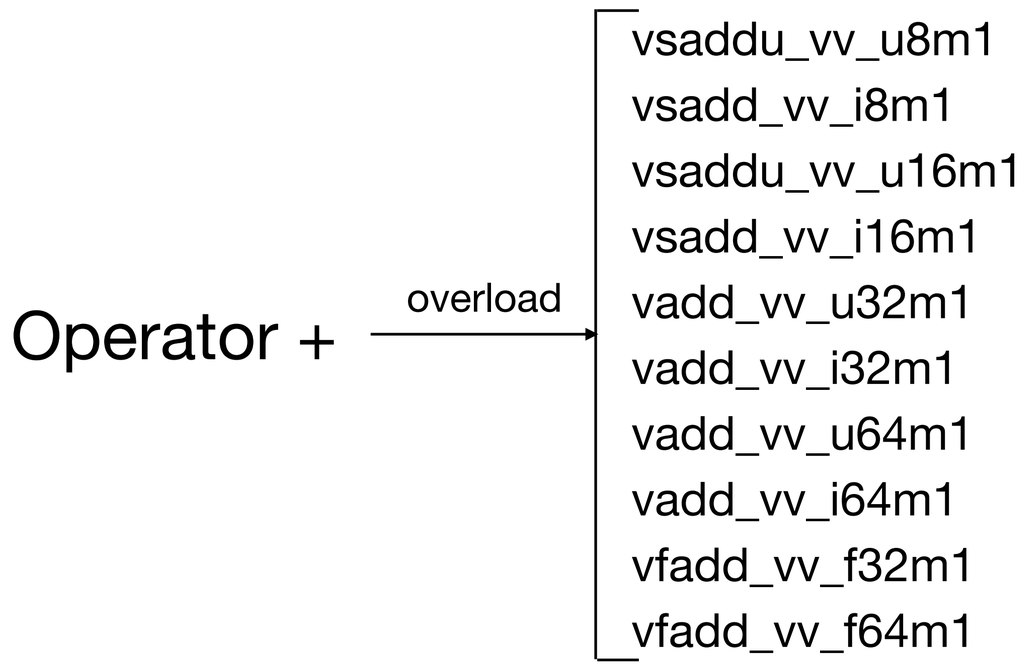
- Comparison:
  @ref operator >(const v_reg &a, const v_reg &b) ">",
  @ref operator >=(const v_reg &a, const v_reg &b) ">=",
  @ref operator <(const v_reg &a, const v_reg &b) "<",
  @ref operator <=(const v_reg &a, const v_reg &b) "<=",
  @ref operator==(const v_reg &a, const v_reg &b) "==",
  @ref operator !=(const v_reg &a, const v_reg &b) "!="

Implementation

Vector Operations

Arithmetic, bitwise and comparison operations

Element-wise binary and unary operations.



Implementation

Vector Operations

Macro for Overload

```
////////////// Initial ////////////

#define OPENCV_HAL_IMPL_RVV_INIT_INTEGER(_Tpvec, _Tp, suffix1, suffix2) \
inline v_##_Tpvec v_setzero_##suffix1() { return v_##_Tpvec(vzero_##suffix2##m1()); } \
inline v_##_Tpvec v_setall_##suffix1(_Tp v) { return v_##_Tpvec(vmv_v_x_##suffix2##m1(v)); }

OPENCV_HAL_IMPL_RVV_INIT_INTEGER(uint8x16, uchar, u8, u8)
OPENCV_HAL_IMPL_RVV_INIT_INTEGER(int8x16, schar, s8, i8)
OPENCV_HAL_IMPL_RVV_INIT_INTEGER(uint16x8, ushort, u16, u16)
OPENCV_HAL_IMPL_RVV_INIT_INTEGER(int16x8, short, s16, i16)
OPENCV_HAL_IMPL_RVV_INIT_INTEGER(uint32x4, unsigned, u32, u32)
OPENCV_HAL_IMPL_RVV_INIT_INTEGER(int32x4, int, s32, i32)
OPENCV_HAL_IMPL_RVV_INIT_INTEGER(uint64x2, uint64, u64, u64)
OPENCV_HAL_IMPL_RVV_INIT_INTEGER(int64x2, int64, s64, i64)
```

Implementation

Vector Operations

Some intrinsics cannot match directly

```
inline v_float32x4 v_matmul(const v_float32x4& v, const v_float32x4& m0,
                            const v_float32x4& m1, const v_float32x4& m2,
                            const v_float32x4& m3)
{
    vfloat32m1_t res = vfmul_vf_f32m1(m0, v_extract_n<0>(v));
    res = vfmacc_vf_f32m1(res, v_extract_n<1>(v), m1);
    res = vfmacc_vf_f32m1(res, v_extract_n<2>(v), m2);
    res = vfmacc_vf_f32m1(res, v_extract_n<3>(v), m3);
    return v_float32x4(res);
}
```

Implementation

Vector Operations

Some compiler-unsupported native vector types and intrinsics

```
struct vuint8mf2_t
{
    uchar val[8] = {0};
    vuint8mf2_t() {}
    vuint8mf2_t(const uchar* ptr)
    {
        for (int i = 0; i < 8; ++i)
        {
            val[i] = ptr[i];
        }
    }
};
struct vint8mf2_t
{
    schar val[8] = {0};
    vint8mf2_t() {}
    vint8mf2_t(const schar* ptr)
    {
        for (int i = 0; i < 8; ++i)
        {
            val[i] = ptr[i];
        }
    }
};
```

```
#define OPENCV_HAL_IMPL_RVV_NATIVE_WCVT(_Tpwvec, _Tpvec, _wTp, wcvt, suffix, width, n) \
inline _Tpwvec wcvt (_Tpvec v) \
{ \
    _wTp tmp[n]; \
    for (int i = 0; i < n; ++i) \
    { \
        tmp[i] = (_wTp)v.val[i]; \
    } \
    vsetvlmax_e##width##m1(); \
    return vle##width##_v_##suffix##m1(tmp); \
}

OPENCV_HAL_IMPL_RVV_NATIVE_WCVT(vuint16m1_t, vuint8mf2_t, ushort, vwcvtu_x_x_v_u16m1, u16, 16, 8)
OPENCV_HAL_IMPL_RVV_NATIVE_WCVT(vint16m1_t, vint8mf2_t, short, vwcvt_x_x_v_i16m1, i16, 16, 8)
OPENCV_HAL_IMPL_RVV_NATIVE_WCVT(vuint32m1_t, vuint16mf2_t, unsigned, vwcvtu_x_x_v_u32m1, u32, 32, 4)
OPENCV_HAL_IMPL_RVV_NATIVE_WCVT(vint32m1_t, vint16mf2_t, int, vwcvt_x_x_v_i32m1, i32, 32, 4)
OPENCV_HAL_IMPL_RVV_NATIVE_WCVT(vuint64m1_t, vuint32mf2_t, uint64, vwcvtu_x_x_v_u64m1, u64, 64, 2)
OPENCV_HAL_IMPL_RVV_NATIVE_WCVT(vint64m1_t, vint32mf2_t, int64, vwcvt_x_x_v_i64m1, i64, 64, 2)
```

Build and Test

Compiler and toolchain Build

- riscv-gnu-toolchain build:

```
git clone https://github.com/riscv/riscv-gnu-toolchain
cd riscv-gnu-toolchain
git checkout rvv-intrinsic
git submodule update --init --recursive
./configure --prefix=/opt/RISCV --with-arch=rv64gcv_zfh --with-abi=lp64d
make linux build-qemu –j$(nproc)
```

- rvv-llvm build:

```
git clone https://github.com/isrc-cas/rvv-llvm.git
cd /opt
mkdir rvv-llvm && cd rvv-llvm
cmake -DLLVM_TARGETS_TO_BUILD="X86;RISCV" -DLLVM_ENABLE_PROJECTS=clang
-G "Unix Makefiles" ../../rvv-llvm/llvm
make –j$(nproc)
```

Build and Test

OpenCV Build for RISC-V

- Use riscv-gnu-toolchain as compiler:

```
git clone https://github.com/opencv/opencv
cd opencv
git checkout rvv
mkdir build && cd build
cmake -DCMAKE_TOOLCHAIN_FILE=../platforms/linux/riscv64-gcc.toolchain.cmake ../
make -j$(nproc)
```

- Use Clang-LLVM as compiler:

```
git clone https://github.com/opencv/opencv
cd opencv
git checkout rvv
mkdir build && cd build
cmake -DCMAKE_TOOLCHAIN_FILE=../platforms/linux/riscv64-clang.toolchain.cmake ../
make -j$(nproc)
```

Build and Test

HAL testing

- Use QEMU simulator for test:

```
/opt/RISCV/bin/qemu-riscv64 -cpu rv64,x-v=true opencv/build/bin/opencv_test_core –
gtest_filter=hal*
```

- Result:

```
[----------] Global test environment tear-down
[ SKIPSTAT ] 3 tests skipped
[ SKIPSTAT ] TAG='skip_other' skip 3 tests
[==========] 23 tests from 3 test cases ran. (140 ms total)
[  PASSED  ] 23 tests.
```

All HAL tests passed

Build and Test

HAL testing

- Use QEMU simulator for test:

```
/opt/RISCV/bin/qemu-riscv64 -cpu rv64,x-v=true opencv/build/bin/opencv_test_core
```

- Result:

```
[----------] Global test environment tear-down
[ SKIPSTAT ] 11 tests skipped
[ SKIPSTAT ] TAG='mem_6gb' skip 1 tests
[ SKIPSTAT ] TAG='skip_other' skip 10 tests
[==========] 11493 tests from 244 test cases ran. (3237199 ms total)
[  PASSED  ] 11472 tests.
```

11472 core tests passed

Future Work

Current Issues

In-memory implementation for vector types

```
struct v_uint8x16
{
    typedef uchar lane_type;
    enum { nlanes = 16 };

    v_uint8x16() {}
    explicit v_uint8x16(vuint8m1_t v)
    {
        vse8_v_u8m1(val, v);
    }
    v_uint8x16(uchar v0, uchar v1, uchar v2, uchar v3, uchar v4, uchar v5, uchar v6, uchar v7,
               uchar v8, uchar v9, uchar v10, uchar v11, uchar v12, uchar v13, uchar v14, uchar v15)
    {
        uchar v[] = {v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15};
        for (int i = 0; i < nlanes; ++i)
        {
            val[i] = v[i];
        }
    }
    operator vuint8m1_t() const
    {
        return vle8_v_u8m1(val);
    }
    uchar get0() const
    {
        return val[0];
    }

    uchar val[16];
};
```

Future Work

Current Issues

In-memory implementation for vector types

- RVV vectors are scalable. The size of RVV vector type is unknown at compilation time. VLEN is determined at runtime.
- https://github.com/riscv/riscv-gnu-toolchain/issues/701
- Reading and writing memory will affect the computational efficiency.
- Solutions:
  - A new framework of Wide Universal Intrinsics to fit vector length agnostic architectures.
  - Non-scalable support for RVV from compiler side.

Future Work

Summary

- Achievements:
  - Added RISC-V backend for OpenCV.
  - Implemented all the Wide Universal Intrinsics based on RVV.
  - Successfully built OpenCV for target RISC-V with GCC/Clang-LLVM.
  - Passed all of the HAL accuracy tests.
  - Passed 11472 core tests.

- Future Work:
  - Pass all the core tests.
  - Performance tests and optimization.

- Community support:
  - RISC-V “V” Extension Specification
  - RVV Intrinsic Specification
  - Compiler support:
    - riscv-gnu-toolchain
    - rvv-llvm
  - Hardware support:
    - Development board that supports RISC-V “V” Extension.

References

GSoC Project: https://summerofcode.withgoogle.com/archive/2020/projects/4805294711898112/

Wide Universal Intrinsic: https://docs.opencv.org/master/df/d91/group__core__hal__intrin.html

OpenCV repository: https://github.com/opencv/opencv

RISC-V ISA specification: https://github.com/riscv/riscv-isa-manual

RISC-V “V” extension specification: https://github.com/riscv/riscv-v-spec

RVV Intrinsic specification: https://github.com/riscv/rvv-intrinsic-doc

RISC-V GNU toolchain: https://github.com/riscv/riscv-gnu-toolchain

Rvv-llvm from PLCT Group: https://github.com/isrc-cas/rvv-llvm

In-memory Issue details: https://github.com/riscv/riscv-gnu-toolchain/issues/701

C910 implementation for WUI: https://github.com/damonyu1989/opencv/tree/master-riscv

Thanks!