RISC-V 向量扩展在Clang/LLVM中的支持

智能软件研究中心 王鹏
PLCT实验室：邢明杰、吴伟、张尹、张章、陈影、陆旭凡

2020/06/30

CONTENT

目录

1. RISC-V向量扩展的背景介绍

RISC-V是什么?

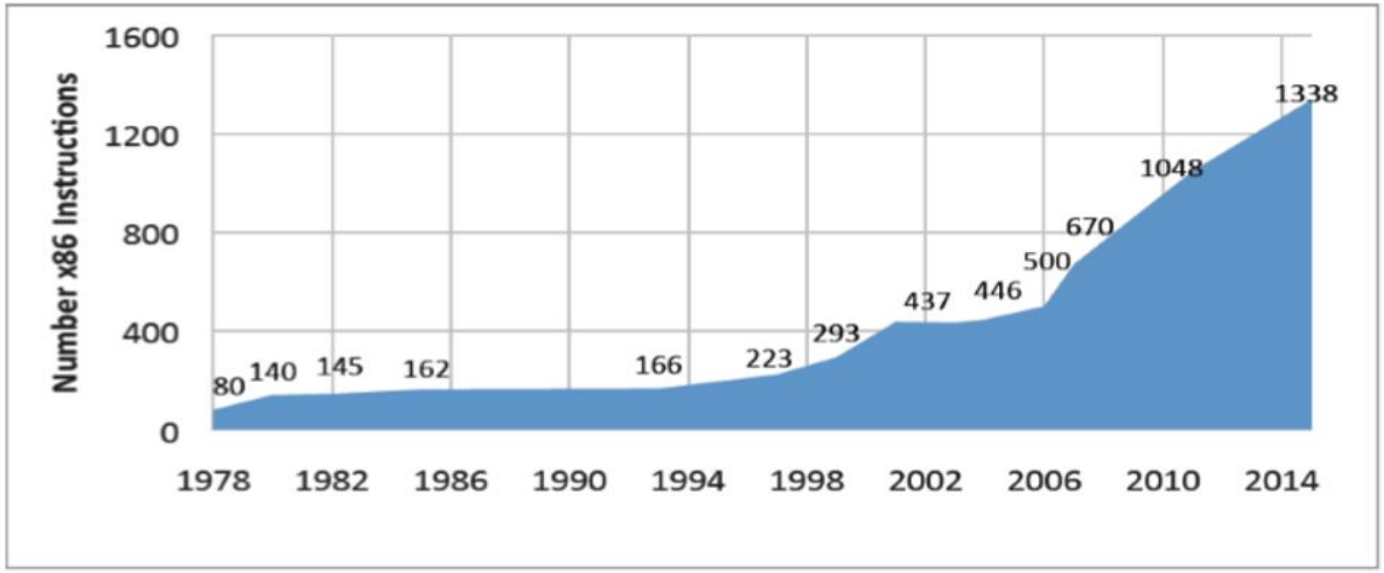
- RISC-V是2011年诞生的一个开源的指令集架构（ISA），它属于一个开放的，非营利性质的基金会，未来不会受任何单一公司的决定的影响。
- RISC-V基金会的目标是保持RISC-V的稳定性，并力图让它如同Linux操作系统一样受欢迎，目前已经有很多的与RISC-V基金会合作的公司机构。

https://riscv.org

1. RISC-V向量扩展的背景介绍

模块化的指令集架构

- x86指令集自诞生以来指令数量的增长。



- 这个增长的很大一部分是因为x86 ISA依赖于SIMD指令来实现数据级并行。

1. RISC-V向量扩展的背景介绍

模块化的指令集架构

基础模块

扩展模块

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*

“IMAFD”模块一般被统称为“G”
例如：RV32GCV

“V”模块
即向量扩展模块

1. RISC-V向量扩展的背景介绍

RISC-V 向量扩展是什么?

- RISC-V向量扩展指令标准草案从2018年开始提出，至今已有多个版本的迭代。中科院软件所智能软件研究中心从2019年10月份开始提供RVV在LLVM上的支持，目前已经先后实现了对v0.7.1、v0.8和v0.9的支持，并且在GitHub上开放了源码。

RISC-V "V" Vector Extension

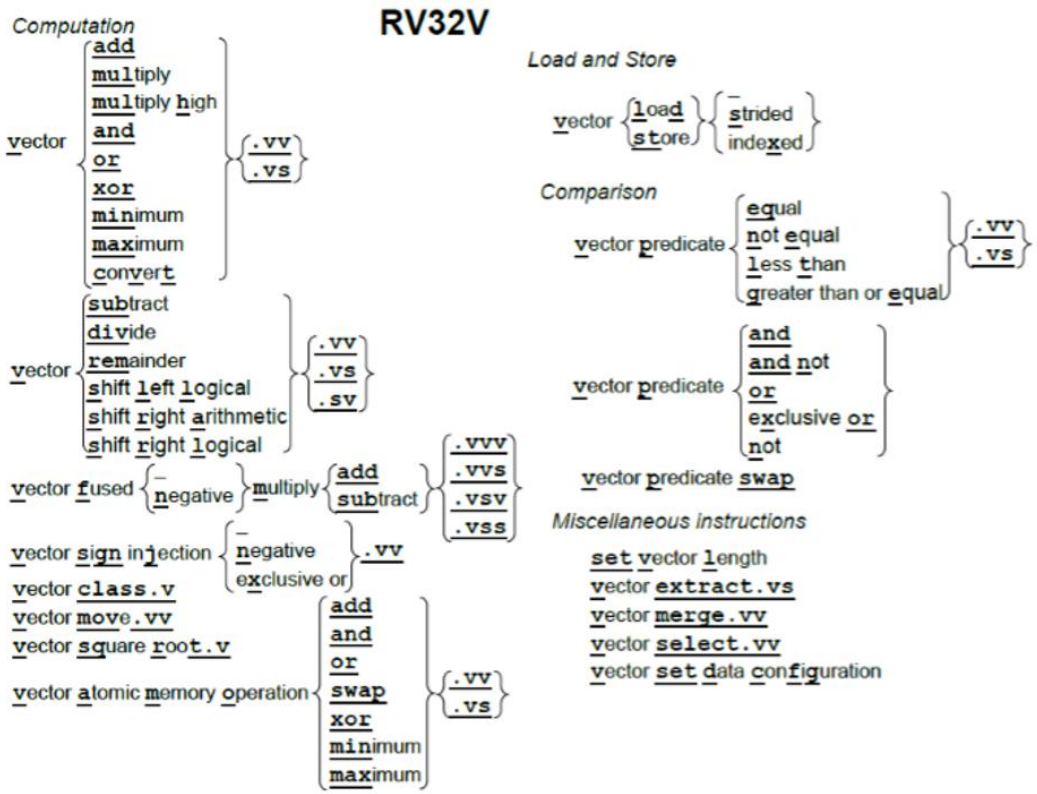
Version 0.9

https://github.com/riscv/riscv-v-spec/releases/tag/0.9

1. RISC-V向量扩展的背景介绍

RISC-V向量扩展简介

- RISC-V向量扩展是RISC-V指令集的标准扩展模块之一(简称RVV）。
- 它主要为基础指令集添加了向量寄存器和各类向量指令，使得用户可以使用向量化来优化和加速程序代码。



https://github.com/riscv/riscv-v-spec/

1. RISC-V向量扩展的背景介绍

为什么使用向量?

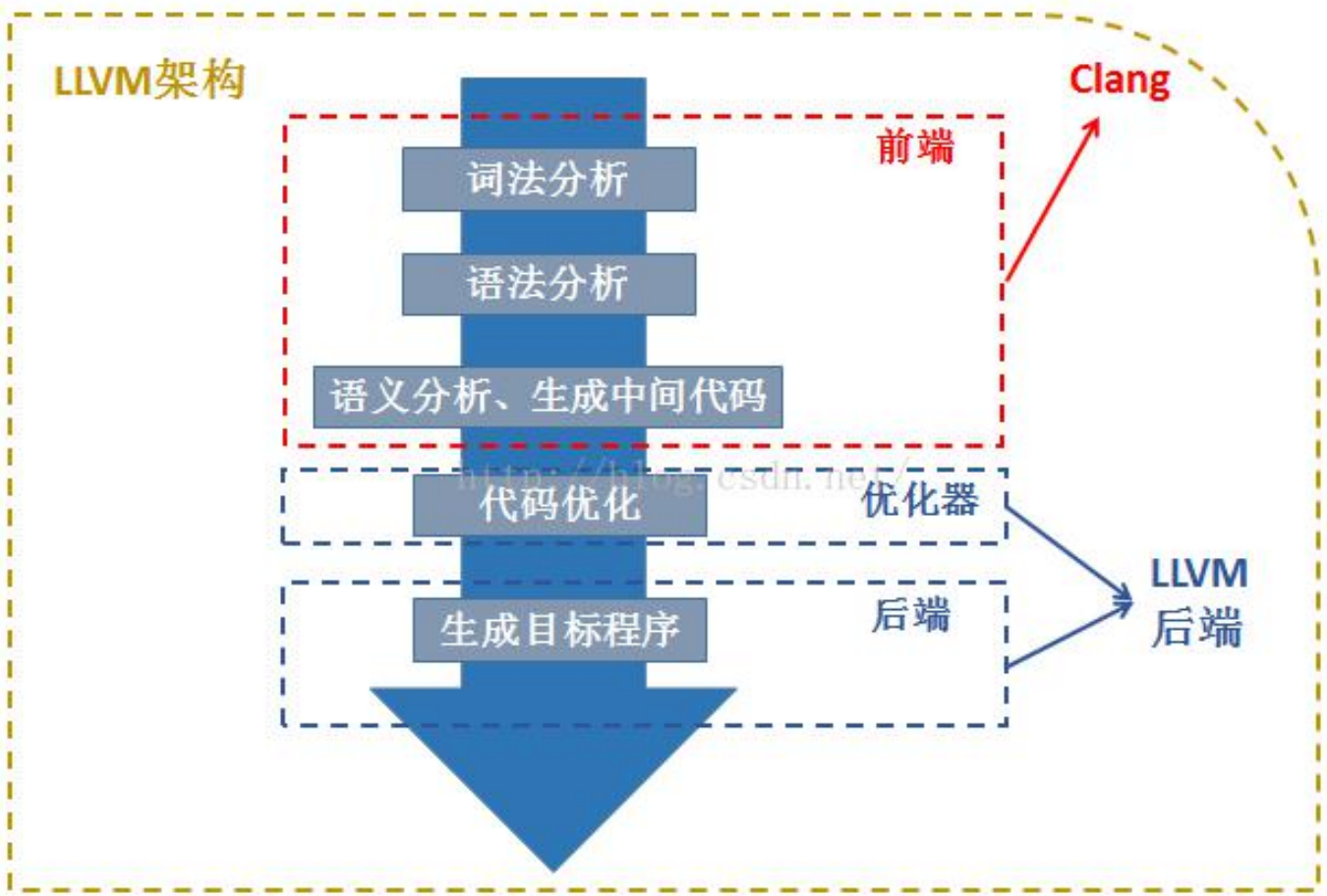
- 当存在大量数据可供应用程序同时计算时，我们称之为数据级并行性。
- 最著名的数据级并行架构是单指令多数据(SIMD，Single Instruction Multiple Data)。
- 由于 SIMD ISA 属于增量设计阵营的 一员，并且操作码指定了数据宽度，因此扩展 SIMD 寄存器也就意味着要同时扩展 SIMD 指令集。
- 将 SIMD 寄存器宽度和 SIMD 指令数量翻倍的后续演进步骤都让 ISA 走上了复杂度逐渐提升的道路。
- 一个更优雅的利用数据级并行性的方案是采用向量架构。

CONTENT

目录

2. RISC-V向量扩展的实现

Clang/LLVM编译器



2. RISC-V向量扩展的实现

Clang/LLVM编译器

我们可以将Clang/LLVM编译器分为前端、中端和后端三个阶段：

- 前端主要处理语言相关部分，词法语法分析、语义分析、构建抽象语法树、生成LLVM IR；
- 中端基于LLVM IR进行各种代码分析和变换；
- 后端主要负责体系结构代码生成。

用户使用RISC-V向量指令编程可以有三种形式：

- Q1 是直接编写汇编代码，
- Q2 是在C/C++程序中使用Intrinsic函数，
- Q3 是通过优化技术将C/C++程序自动转换成向量指令。

2. RISC-V向量扩展的实现

向量指令编程实现

A1：汇编器的支持。在LLVM后端MC层实现。我们在LLVM的RISCV后端目录下新增了相应的目标描述文件，使用TableGen语言来描述向量寄存器，向量指令格式和指令信息，并在汇编/反汇编相关子目录AsmParser/Disassembler下面实现对特定操作数的处理。

A2：编译器的支持。在前端，需要实现向量builtin函数和向量数据类型，并将这些builtin函数转换成LLVM intrinsic函数调用，在后端代码生成阶段，需要将这些intrinsic函数调用转换成具体的向量指令。RVV属于SVE（Scalable Vector Extension），向量大小是不固定的，这种灵活性会导致在编译器中的实现要比汇编指令支持复杂很多。

A3：在编译器中端实现自动向量化的优化支持。我们将在未来开展这部分工作

2.1 RISC-V 向量扩展的特点

变长向量

- RISC-V 不预设向量长度
- 具体实现可以自主选择提供多长的向量
- 应用程序在运行时自主选择需要的长度

可动态调整向量类型（EEW, EMUL）

- RVV可以在编译时动态调整向量类型参数以应对不同的数据类型和数据大小的输入。

2.1 RISC-V 向量扩展的特点

混合类型编程

- RISC-V 允许在一条指令中使用不同长度的向量， 其中短向量会自动扩宽为长向量。

  例如：

  vadd $v1_{i8}$, $v2_{i8}$ -> $v0_{i16}$， v1 将被提升为 i16 向量。

  vadd $v1_{i8}$, $v2_{i64}$ -> v_{i64}， v1 将被提升为 i64 向量。

高度可扩展

- RVV 指令可针对具体领域进行扩展， 比如矩阵计算、DSP、机器学习和图像处理。

2.1 RISC-V 向量扩展的特点

对向量化的丰富支持

1. 几乎每条指令都支持 Mask口语解释mask
2. 各种访存模式（linear or strided loads and stores, scatters and gathers)
3. 各种规约操作（sum, min/max, and/for...）
4. 向量正交运算操作
5. 通过数据依赖结束的fault-only-first loads循环指令

支持定点数和16位浮点数

2.2 RISC-V向量扩展intrinsics简介

RISC-V Vector Intrinsics

- Intrinsics一般是指高级编程语言中的低级汇编语言的接口。
- EPI、SiPearl和SiFive联合发布了一份RISC-V “V”(向量)扩展的Intrinsics的规范，并且已经被RISC-V采纳为标准规范。
- 这份Intrinsics的API的目标是让C/C++可以访问所有的RISC-V "V"(向量)扩展的指令。
- https://github.com/riscv/rvv-intrinsic-doc

2.2 RISC-V向量扩展intrinsics简介

- 对于Intrinsic，我们是基于rvv-intrinsic-doc仓库下的RFC进行实现。目前已经初步实现了向量类型系统，以及一些向量intrinsic函数，可以跑通自己编写的一些简单测试用例。

- 整个项目是基于LLVM上游仓库开发，会在一定时间点上与上游代码进行合并，我们最初的代码是基于Robin Kruppe的实现进行开发的。并且在开发过程中借鉴了EPI的代码实现。

- https://github.com/rkruppe/rvv-llvm.git

2.2 RISC-V向量扩展intrinsics简介

类型系统

数据类型 (Data Types)

将SEW和LMUL编码为数据类型。SEW≤64的数据类型如下。

$Types$	$LMUL=1$	$LMUL=2$	$LMUL=4$	$LMUL=8$
$int64\_t$	$vint64m1\_t$	$vint64m2\_t$	$vint64m4\_t$	$vint64m8\_t$
$uint64\_t$	$vuint64m1\_t$	$vuint64m2\_t$	$vuint64m4\_t$	$vuint64m8\_t$
$int32\_t$	$vint32m1\_t$	$vint32m2\_t$	$vint32m4\_t$	$vint32m8\_t$
$uint32\_t$	$vuint32m1\_t$	$vuint32m2\_t$	$vuint32m4\_t$	$vuint32m8\_t$
$int16\_t$	$vint16m1\_t$	$vint16m2\_t$	$vint16m4\_t$	$vint16m8\_t$
$uint16\_t$	$vuint16m1\_t$	$vuint16m2\_t$	$vuint16m4\_t$	$vuint16m8\_t$
$int8\_t$	$vint8m1\_t$	$vint8m2\_t$	$vint8m4\_t$	$vint8m8\_t$
$uint8\_t$	$vuint8m1\_t$	$vuint8m2\_t$	$vuint8m4\_t$	$vuint8m8\_t$
$vfloat64$	$vfloat64m1\_t$	$vfloat64m2\_t$	$vfloat64m4\_t$	$vfloat64m8\_t$
$vfloat32$	$vfloat32m1\_t$	$vfloat32m2\_t$	$vfloat32m4\_t$	$vfloat32m8\_t$
$vfloat16$	$vfloat16m1\_t$	$vfloat16m2\_t$	$vfloat16m4\_t$	$vfloat16m8\_t$

掩码类型 (Data Types)

将MLEN编码为掩码类型。SEW≤64的掩码类型如下。

$Types$	$MLEN=1$	$MLEN=2$	$MLEN=4$	$MLEN=8$	$MLEN=16$	$MLEN=32$	$MLEN=64$
$bool$	$vbool1\_t$	$vbool2\_t$	$vbool4\_t$	$vbool8\_t$	$vbool16\_t$	$vbool32\_t$	$vbool64\_t$

2.2 RISC-V向量扩展intrinsics简介

通用命名规则 (Naming Rules)

```
INTRINSIC ::= MNEMONIC '_' RET_TYPE
MNEMONIC ::= Instruction name in v-ext specification. Replace '.' with '_'.
RET_TYPE ::= SEW LMUL
SEW ::= ( i8 | i16 | i32 | i64 | u8 | u16 | u32 | u64 | f16 | f32 | f64 )
LMUL ::= ( m1 | m2 | m4 | m8 )
```

```
Example:

vadd.vv vd, vs2, vs1:
vint8m1_t vadd_vv_i8m1(vint8m1_t vs2, vint8m1_t vs1)

vwaddu.vv vd, vs2, vs1:
vint16m2_t vwaddu_vv_i16m2(vint8m1_t vs2, vint8m1_t vs1)
```

2.2 RISC-V向量扩展intrinsics简介

例外命名规则 (Exceptions in Naming)

Example:

vsb.v vs3, (rs1):
void vsb_v_i8m1(int8_t *rs1, vint8m1_t vs3);

Example:

vmseq.vv vd, vs2, vs1:
vbool8_t vmseq_vv_i8m1_b8(vint8m1_t vs2, vint8m1_t vs1);
vbool8_t vmseq_vv_i16m2_b8(vint16m2_t vs2, vint16m2_t vs1);

2.2 RISC-V向量扩展intrinsics简介

在C/C++中的使用

```
void saxpy_vec(size_t n, const float a, const float *x, float *y) {
  size_t l;

  vfloat32m8_t vx, vy;

  for (; (l = vsetvl_e32m8(n)) > 0; n -= l) {
    vx = vle_v_f32m8(x);
    x += l;
    vy = vle_v_f32m8(y);
    vy = vfmacc_vf_f32m8(vy, a, vx);
    vse_v_f32m8 (y, vy);
    y += l;
  }
}
```

2.3 RISC-V向量扩展汇编指令支持

- 对于汇编指令，目前我们已经支持最新的稳定版本v0.9，可以通过自己添加的测试用例，gnu工具链binutils中的测试用例（除了EDIV不支持），以及riscv-v-spec仓库中的所有例子。

- 其中针对riscv-v-spec仓库中的例子的测试框架也在github上开放。
- Link: https://github.com/isrc-cas/rvv-benchmark

2.3 RISC-V向量扩展汇编指令支持

```
360 //===----------------------------------------------------
361 // Vector unit CSRs (made-up CSR numbers for now)
362 //===----------------------------------------------------
363 // 0xCC0 is a non-standard read-only user mode CSR
364 def VLCSR : SysReg<"vl", 0xCC0>;
365 def VLMAXCSR : SysReg<"vlmax", 0xCC1>;
366
367 //===----------------------------------------------------
368 // User Vector CSRs
369 //===----------------------------------------------------
370 def : SysReg<"vstart", 0x008>;
371 def : SysReg<"vxsat", 0x009>;
372 def : SysReg<"vxrm", 0x00A>;
373 def : SysReg<"vcsr", 0x00F>;
374 def : SysReg<"vl", 0xC20>;
375 def : SysReg<"vtype", 0xC21>;
```

- https://github.com/isrc-cas/rvv-llvm/llvm/lib/Target/RISCV/RISCVSystemOperands.td

2.3 RISC-V向量扩展汇编指令支持

```
u@u-virtual-machine:~/tools/rvv-llvm$ echo "csrrs t1, vcsr, zero" | llvm-mc -triple=riscv32 -mattr=+v -show-encoding
        .text
        csrr    t1, vcsr                # encoding: [0x73,0x23,0xf0,0x00]
u@u-virtual-machine:~/tools/rvv-llvm$ echo "csrrs t2, vcsr, zero" | llvm-mc -triple=riscv32 -mattr=+v -show-encoding
        .text
        csrr    t2, vcsr                # encoding: [0xf3,0x23,0xf0,0x00]
u@u-virtual-machine:~/tools/rvv-llvm$ echo "0x73,0x23,0xf0,0x00" | llvm-mc -triple=riscv32 -mattr=+v -disassemble -show-in
st
        .text
        csrr    t1, vcsr                # <MCInst #368 CSRRS
                                        #  <MCOperand Reg:40>
                                        #  <MCOperand Imm:15>
                                        #  <MCOperand Reg:34>>
u@u-virtual-machine:~/tools/rvv-llvm$ echo "0xf3,0x23,0xf0,0x00" | llvm-mc -triple=riscv32 -mattr=+v -disassemble -show-in
st
        .text
        csrr    t2, vcsr                # <MCInst #368 CSRRS
                                        #  <MCOperand Reg:41>
                                        #  <MCOperand Imm:15>
                                        #  <MCOperand Reg:34>>
```

- https://github.com/isrc-cas/rvv-llvm/llvm/test/MC/RISCV/user-csr-names.s

2.3 RISC-V向量扩展汇编指令支持

```
u@u-virtual-machine:~/tools/rvv-llvm$ ./build/bin/llvm-lit -v ./llvm/test/MC/RISCV/user-csr-names.s
-- Testing: 1 tests, 1 workers --
PASS: LLVM :: MC/RISCV/user-csr-names.s (1 of 1)

Testing Time: 0.43s
  Expected Passes: 1
```

- https://github.com/isrc-cas/rvv-llvm/llvm/test/MC/RISCV目录
- user-csr-names.s
- rvv-valid.s
- rvv-mask-valid.s
- rvv-pseudo-valid.s
- rvv-pseudo-mask-valid.s

CONTENT

目录

3. RISC-V向量扩展的支持现状

GCC的支持现状

目前，GCC工具链已经较完整支持了RISC-V “V”向量扩展的汇编，并且持续在跟进向量扩展的最新版本。Intrinsics尚未支持。

GitHub Link:
binutils-gdb: https://github.com/riscv/riscv-binutils-gdb
gnu-toolchain: https://github.com/riscv/riscv-gnu-toolchain

RISC-V的这两个gcc相关仓库中都有相应的**rvv-0.7.x**，**rvv-0.8.x**和**rvv-0.9.x**分支。对应支持了相应版本规范的RISC-V向量扩展。binutils-gdb最新推出了**rvv-1.0.x**

3. RISC-V向量扩展的支持现状

GNU编译工具链针对向量扩展的使用注意事项

在安装riscv-gnu-toochain时，如果想要使用RISC-V向量扩展，需要在clone下github仓库后切换至rvv分支（最新分支为**rvv-1.0.x**）。并且在configure时使用**--with-arch**选项添加v后缀（例如：**--with-arch=rv32gcv**）。

The build defaults to targetting RV64GC (64-bit), even on a 32-bit build environment. To build the 32-bit RV32GC toolchain, use:

```
./configure --prefix=/opt/riscv --with-arch=rv32gc --with-abi=ilp32d
make linux
```

3. RISC-V向量扩展的支持现状

3.1 PLCT实验室在LLVM上的支持

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

- 在该仓库的主分支**rvv-iscas**分支中，目前已经完成了对RSIC-V向量扩展的最新版本v0.9版的汇编支持。
- 实现了RSIC-V向量扩展的部分intrinsics支持。
- 由于目前Intrinsics规范文档已经更新至最新的v0.9版本，后续intrinsics的完整支持也会在主分支上基于v0.9版本的汇编进行实现。

3. RISC-V向量扩展的支持现状

3.2 PLCT实验室实现的rvv-benchmark

Vector Assembly Code Example

Assembly	Descriptiion	Status
memcpy.s	memory copy example	☑
strlen.	return string length example	☑
strcpy.	copy string example	☑
strncpy.s	copy fixed string of size n	☑
saxpy.s	y[i] = a * x[i] + y[i]	☑
sgemm.S	c[m][n] += a[m][k]*b[k][n]	☑
vvaddint32.s	vector-vector add example	☑

GitHub Link: https://github.com/isrc-cas/rvv-benchmark

3. RISC-V向量扩展的支持现状

3.2 PLCT实验室实现的rvv-benchmark

```
objects = main.c memcpy.s sgemm.S strlen.s vvaddint32.s saxpy.s strcpy.s strncpy.s

target = rvv1

$(target) : $(objects)
        clang  --target=riscv64-unknown-elf -march=rv64gv --sysroot=/home/u/tools/riscv64/install/riscv64-unknown-elf  --gcc-toolchain=/home/u/tools/riscv64/install -o rvv1 $(objects)
        spike --isa=rv64gv pk $(target)

clean :
        rm -f $(objects) $(target)
```

3. RISC-V向量扩展的支持现状

3.2 PLCT实验室实现的rvv-benchmark

```
414 00000000000106ec <memcpy>:
415    106ec:       00050693                mv      a3,a0
416
417 00000000000106f0 <loop>:
418    106f0:       003672d7                vsetvli t0,a2,e8,m8,ta,ma,d1
419    106f4:       0205f007                vle64.v v0,(a1)
420    106f8:       005585b3                add     a1,a1,t0
421    106fc:       40560633                sub     a2,a2,t0
422    10700:       0206f027                vse64.v v0,(a3)
423    10704:       005686b3                add     a3,a3,t0
424    10708:       fe0614e3                bnez    a2,106f0 <loop>
425    1070c:       00008067                ret
426
```

- $riscv64-unknown-elf-objdump -d elf>& elf.dump

3. RISC-V向量扩展的支持现状

3.3 RVV类型系统

```
13 #define RISCV_VECTOR_NAME(LMUL, SEW, NAME) v##NAME##SEW##m##LMUL##_t
14
15 #define RISCV_VECTOR_TYPE(LMUL, SEW, Kind, NAME)
       \
16   typedef __attribute((riscv_vector_type(LMUL, SEW)))
       \
17       Kind RISCV_VECTOR_NAME(LMUL, SEW, NAME)
18
19 #define RISCV_VECTOR_TUPLE_NAME(LMUL, SEW, NAME, NR)
       \
20   v##NAME##SEW##m##LMUL##x##NR##_t
21
22 #define RISCV_VECTOR_TUPLE_TYPE_1(LMUL, SEW, NAME)
       \
23   typedef struct {
       \
24     RISCV_VECTOR_NAME(LMUL, SEW, NAME) v1;
       \
25   } RISCV_VECTOR_TUPLE_NAME(LMUL, SEW, NAME, 1)
26
```

https://github.com/isrc-cas/rvv-llvm/clang/lib/Headers/riscv_vector.h

3. RISC-V向量扩展的支持现状

3.4 使用TableGen自动生成riscv_vector.h

```
class RISCVBuiltin<string suffix, string prototype,
                   string type_range>
{
  string Suffix = suffix;
  string Prototype = prototype;
  string TypeRange = type_range;

  bit HasMask = 1;;
  bit HasVL = 1;
  bit HasSideEffects = 0;
  list<int> LMUL = [1, 2, 4, 8];
  code ManualCodegen = [{ CGM.getIntrinsic(E, "RISCV builtin");
                          return llvm::UndefValue::get(ResultType); }];
}

//===----------------------------------------------------------------------===//
// Builtin definitions.
//===----------------------------------------------------------------------===//
let HasVL = 0, HasMask = 0, HasSideEffects = 1, LMUL = [1],
    ManualCodegen = [{}] in
{
  def vsetvl : RISCVBuiltin<"", "iii", "n">;
```

https://github.com/isrc-cas/rvv-llvm/clang/include/clang/Basic/riscv_vector.td

CONTENT

目录

4. RISC-V向量扩展的未来工作

4.1 完成Intrinsic的支持

后续我们将继续完善对intrinsic的支持，并且会在OpenCV中使用RVV intrinsics来实现基于RISC-V向量指令的Wide Universal Intrinsics，使得OpenCV在RISC-V平台上可以使用RISC-V向量扩展来加速运算。

4. RISC-V向量扩展的未来工作

4.2 自动向量化

所谓的向量化，简单理解，就是使用高级的向量化SIMD指令（如SSE）优化程序，属于数据并行的范畴。

知道了向量化的目标是生成SIMD指令，那么很显然，要对代码进行向量化，第一是依靠编译器来生成这些指令；第二是使用汇编或Intrinsics函数。Intel编译器中，利用其自动向量分析器（auto-vectorizer）对代码进行分析并生成SIMD指令。

编译器的自动向量化，简单理解，就是编译器对代码进行一些处理从而生成SIMD指令，一个最常见的可以进行向量化的例子就是for循环，这里简单的理解一下，如下代码片段：

4. RISC-V向量扩展的未来工作

4.2 自动向量化

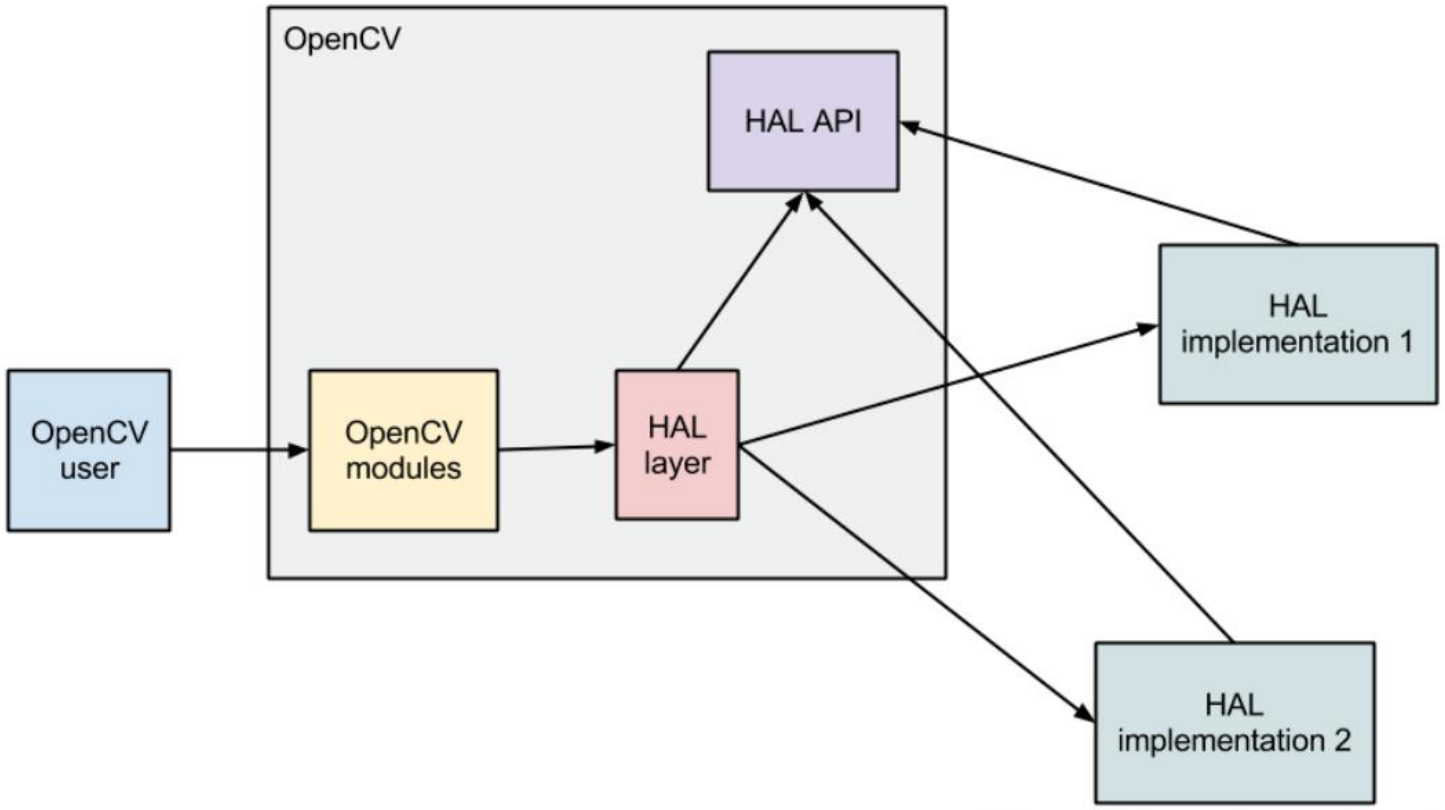
```
1 #define  N  4*10
2
3 float a[N], b[N], c[N];
4
5 for(int i = 0;i < N; i ++)
6 {
7         a[i] = b[i] + c[i];
8 }
9
```

对于上面的代码，假设float类型是4byte，即32bit。如果使用“传统”的指令，这个for循环至少需要经过40次浮点加法运算。那么，如果使用SSE指令（128bit暂存器），那么一条指令能同时计算4个float，所以，上面的for循环可以只适用10次浮点加法运算就完成了。

4. RISC-V向量扩展的未来工作

4.3 OpenCV中的应用

OpenCV在其硬件加速层（HAL）定义了Wide Universal Intrinsics。



4. RISC-V向量扩展的未来工作

4.3 OpenCV中的应用

OpenCV的硬件加速层全称是OpenCV Hardware Acceleration Layer (HAL)，一般来说，硬件厂商或开发人员可能会根据自己的实际情况，开发出独立于OpenCV的运算函数，来支持硬件优化，加快运算速度。这些功能性模块有可能是闭源的。为了实现顺利对接，OpenCV提供了一个简单的接口模块，当各厂家需要开发自己的运算模块时，只要实现这些接口即可。

4. RISC-V向量扩展的未来工作

4.3 OpenCV中的应用

Wide Universal Intrinsics是一种OpenCV中通用的向量内部函数，它对应有不同后端的实现，使得OpenCV可以在不同平台上运行时选择使用该平台支持的SIMD和向量指令集来加速运算。

目前OpenCV的Wide Universal Intrinsics已经有很多基于x86和ARM平台向量扩展的实现，例如AVX，SSE，NEON等。为OpenCV的Wide Universal Intrinsics添加基于RISC-V向量指令的后端实现是我们目前正在进行中的一项工作。这项工作中会使用rvv intrinsics来实现基于RISC-V向量指令的Wide Universal Intrinsics，可以使OpenCV在RISC-V平台上使用RISC-V向量扩展来加速运算。

CONTENT

目录

5. 参考文献

The RISC-V Vector ISA

Wikipedia: RISC-V

Adventures with RISC-V Vectors and LLVM

RISC-V VECTORS KNOW NO LIMITS

Intel® AVX-512 architecture evolution and support in Clang/LLVM

Scalable Vector Extension (SVE) for Armv8-A

RISC-V Vector Extension, v0.9

Berkeley Lectures L17-RISCV-Vectors

Getting Started with LLVM Core Libraries

谢谢大家
欢迎大家加入PLCT实验室

如有任何疑问欢迎与我联系：wangpeng5@iie.ac.cn