RISC-V Vector 及定制指令 gem5 实践

希姆计算 欧阳鑫

gem5 与 RISC-V

RVV ISA及定制指令实现

验证及总结

gem5仿真器是用于计算机系统体系结构（系统级及处理器微架构）研究的模块化平台。

http://www.gem5.org

gem5 提供了什么?

- 多种架构 ISA 实现
- 全面的模型库
  - 多种处理器模型
  - 内存及网络模型，I/O 设备等
- 可以基于模型库快速进行系统建模
- 很容易添加自定义模型
- 活跃的开发者及用户社区

RISC-V in gem5

- RISC-V 初始支持 [A. Roelke, CARRV 2017]
  - RV64GC，单核系统，System call emulation (SE) mode
- 多核系统支持 [CARRV 2018]
- 基础Baremetal Fullsystem及示例配置
- 不支持Linux，Android等操作系统的Fullsystem mode
- 基于riscv-tests的指令测试集

gem5 与 RISC-V

RVV ISA及定制指令实现

验证及总结

RVV ISA 简介

- 32 个向量寄存器, v0-v31
- 操作之前先通过 vsetvl* 特殊指令配置 RVV指令操作的vtype及长度
- 数据类型寄存器 vtype , 用于解释向量寄存器中的向量内容， 如元素位宽等
- 向量长度寄存器 vl , 控制向量指令操作的元素数量
- 向量指令支持 mask 屏蔽一些元素
- 向量内存操作支持：单位步长(Unit-Stride)、跨距(Strided)和索引(Indexed)等寻址模式

在 gem5 中支持 RVV 指令集扩展

- RVV ISA实现
- RVV Registers
- Vector 功能单元或协处理器
- 基于 riscv-tests 的测试集

ISA 实现

- gem5提供了自定义的 ISA DSL，用于生成gem5所需的指令类定义和译码器功能。
- 解析器将从类似C++的代码中提取指令特征

RVV ISA 实现：decode

```
// vector
0x15: decode FUNCT3 {
    // OPIVV
    0x0: decode FUNCT6 {
        format VIntOp {
            0x0: vadd_vv({{
                for (unsigned i = 0; i < eCount; i++) {
                    Vd_vi[i] = Vs1_vi[i] + Vs2_vi[i];
                }
            }});
            0x2: vsub_vv({{
                for (unsigned i = 0; i < eCount; i++) {
                    Vd_vi[i] = Vs1_vi[i] - Vs2_vi[i];
                }
            }});
            0x4: VUIntOp::vminu_vv({{
                for (unsigned i = 0; i < eCount; i++) {
                    Vd_vi[i] = std::min(Vs1_vi[i], Vs2_vi[i]);
                }
            }});
```

bitfield

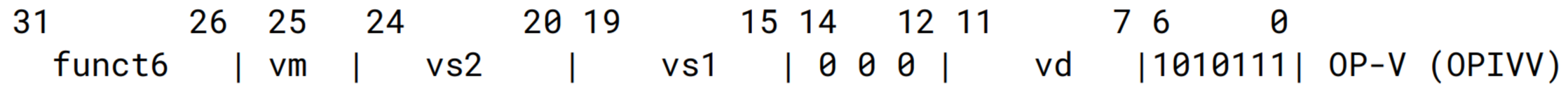
format

code

operand

operand_type

RVV ISA 实现：bitfields




```
// bitfields.isa

def bitfield FUNCT3 <14:12>;
def bitfield FUNCT6 <31:26>;
def bitfield VM <25>;
def bitfield VS2 <24:20>;
def bitfield VS1 <19:15>;
def bitfield VD <11:7>;
def bitfield ZIMM <30:20>;
def bitfield MOP <28:26>;
```

RVV ISA 实现：operands

```
def operand_types {{
    'sb' : 'int8_t',
    'ub' : 'uint8_t',
    ...
    'vi': 'vi',
    'vf': 'vf'
}};
```

```
def operands {{
#General Purpose Integer Reg Operands
    'Rd': ('IntReg', 'xu', 'RD', 'IsInteger', 1),
    'Rs1': ('IntReg', 'xu', 'RS1', 'IsInteger', 2),
    'Rs2': ('IntReg', 'xu', 'RS2', 'IsInteger', 3),
    ...
    'Vd': ('VecReg', 'ud', 'VD', 'IsVector', 1),
    'Vds': ('VecReg', 'ud', 'VD', 'IsVector',1),
    'Vs1': ('VecReg', 'ud', 'VS1', 'IsVector', 2),
    'Vs2': ('VecReg', 'ud', 'VS2', 'IsVector', 3),
    'Vs3': ('VecReg', 'ud', 'VS3', 'IsVector', 4),
    'V0': ('VecReg', 'uh', '0', 'IsVector', 5),
```

RVV ISA 实现：formats

```
def format VIntOp(code, exception_code={{;}}, *opt_flags) {{
    iop = InstObjParams(name, Name, 'VOp', {'code':code,
        'exception_code': exception_code}, opt_flags)
    header_output = BasicDeclare.subst(iop)
    decoder_output = BasicConstructor.subst(iop)
    decode_block = BasicDecode.subst(iop)
    exec_output = VectorIntExecute.subst(iop)
}};
```

```
0x0: decode FUNCT6 {
    format VIntOp {
        0x0: vadd_vv({{
            for (unsigned i = 0; i < eCount; i++) {
                Vd_vi[i] = Vs1_vi[i] + Vs2_vi[i];
            }
        }});
```

```
def template VectorIntExecute {{
    Fault %(class_name)s::execute(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Fault fault = NoFault;

        unsigned eCount = getVecLen(xc->tcBase());
        unsigned eWidth = getVecSew(xc->tcBase());

        switch (eWidth) {
            case 8: {
                using vi = int8_t;
                %(op_decl)s;
                %(op_rd)s;
                if (fault != NoFault) return fault;
                %(code)s;
                if (fault != NoFault) return fault;
                %(op_wb)s;
                break;
            }
            case 16: {
                using vi = int16_t;
                %(op_decl)s;
                %(op_rd)s;
                if (fault != NoFault) return fault;
                %(code)s;
                if (fault != NoFault) return fault;
                %(op_wb)s;
                break;
            }
            case 32: {
```

RVV ISA 实现：自动生成的c++代码

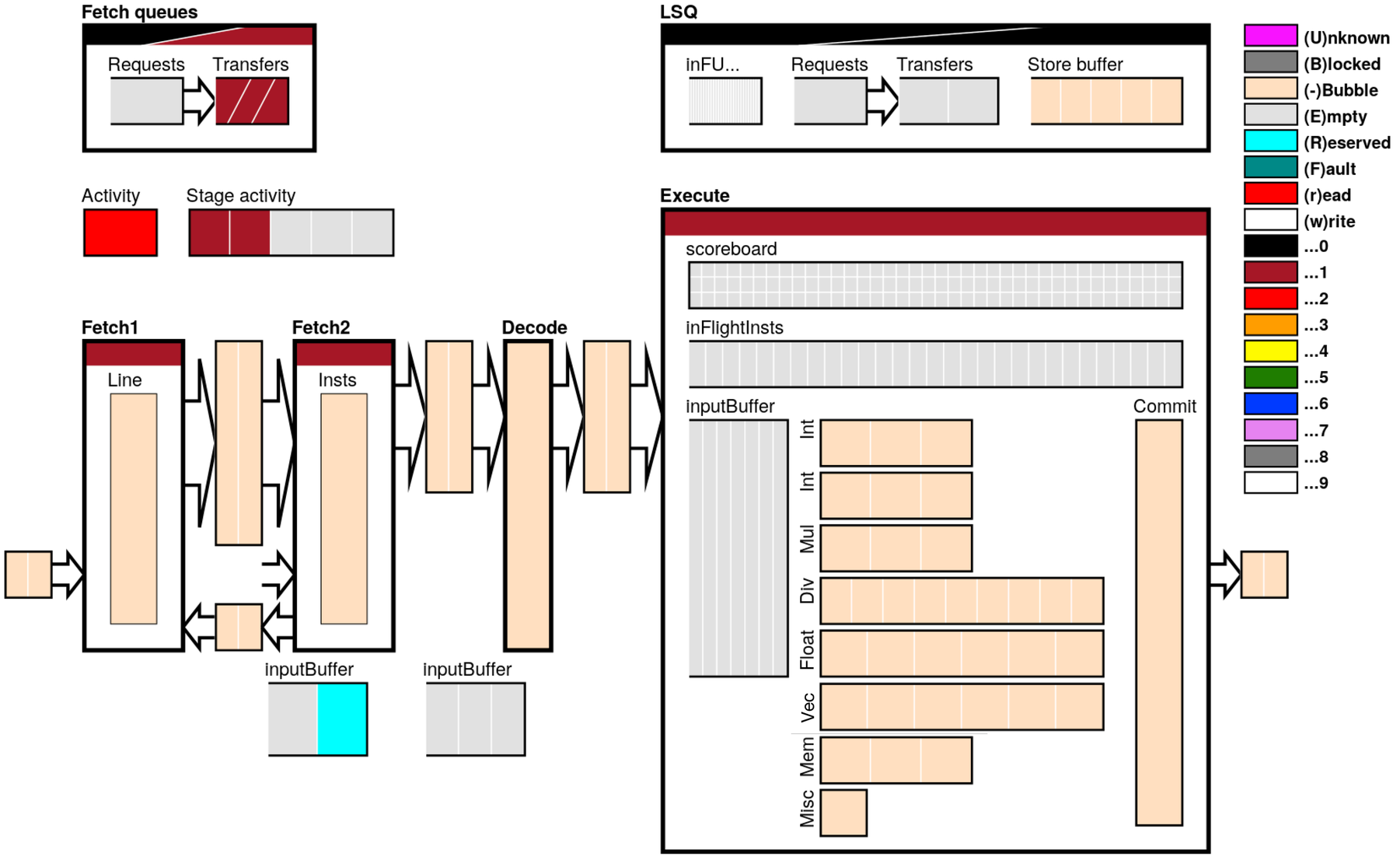
```
case 0x15:
  switch (FUNCT3) {
    case 0x0:
      switch (FUNCT6) {
        case 0x0:
          return new Vadd_vv(machInst);
        break;
        case 0x2:
          return new Vsub_vv(machInst);
        break;

        case 0x4:
          return new Vminu_vv(machInst);
        break;
```

```
Fault Vadd_vv::execute(ExecContext *xc,
    Trace::InstRecord *traceData) const
{
    Fault fault = NoFault;

    ;;
    if (fault != NoFault) return fault;

    unsigned eCount = getVecLen(xc->tcBase());
    unsigned eWidth = getVecSew(xc->tcBase());

    switch (eWidth) {
        case 8: {
            using vi = int8_t;

            TheISA::VecRegContainer& tmp_d0 = xc->getWritableVecRegOperand(this, 0);
            auto Vd = tmp_d0.as<vi>();
            const TheISA::VecRegContainer& tmp_s0 = xc->readVecRegOperand(this, 0);
            auto Vs1 = tmp_s0.as<vi>();
            const TheISA::VecRegContainer& tmp_s1 = xc->readVecRegOperand(this, 1);
            auto Vs2 = tmp_s1.as<vi>();

            if (fault != NoFault) return fault;

            for (unsigned i = 0; i < eCount; i++) {
                Vd[i] = Vs1[i] + Vs2[i];
            }
```

RVV Registers

```
const std::vector<std::string> VecRegNames = {
    "v0", "v1", "v2", "v3",
    "v4", "v5", "v6", "v7",
    "v8", "v9", "v10", "v11",
    "v12", "v13", "v14", "v15",
    "v16", "v17", "v18", "v19",
    "v20", "v21", "v22", "v23",
    "v24", "v25", "v26", "v27",
    "v28", "v29", "v30", "v31"
};

enum CSRIndex {
    ...
    CSR_VL = 0xC20,          // Vector length
    CSR_VTYPE = 0xC21,       // Vector data type register
```

Vector 功能单元（InOrder CPU）



or 对 Vector 协处理器 建模

RISC-V 定制指令

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 ($> 32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

Table 24.1: RISC-V base opcode map, inst[1:0]=11

定制指令实现

```
// custom-0
0x02: Unknown::unknown();

// custom-1
0x0a: Unknown::unknown();

// custom-2
0x16: decode FUNCT3 {
    format VIntOp {
        0x0: vmod_vv({{
            for (unsigned i = 0; i < eCount; i++) {
                Vd_vi[i] = Vs1_vi[i] % Vs2_vi[i];
            }
        }});
    }
}
// custom-3
0x1e: decode FUNCT6 {
```

gem5 基础知识

RVV ISA及定制指令实现

验证及总结

基于 riscv-tests 的RVV指令测试集

- 添加 RVV 指令的cases
- 使用 numpy 生成 golden 数据
- 使用 gem5 执行 cases

功能验证

- 我们可以在如下平台上，使用上述测试集进行功能验证
  - gem5，AtomicCPU/MinorCPU
  - Spike
  - FPGA原型
  - 实际支持RVV指令的处理器
- 保证测试结果及内存数据均为一致

时钟准确度验证

- gem5提供的Stat统计功能可以得到cycle数，cache命中率等关键指标，用于分析模型的时钟准确度
- gem5只提供了通用CPU模型，通过使用组件库构建目标系统并调整各模型组件的参数，可以快速得到一个原型系统
- 更精确的系统模型需要对CPU流水线、内存及网络结构进行更详细的建模及参数调整

谢谢