UC TECHIP
优矽科技

RISC-V Vector性能分析

叶锡聪

2020.07.18

UC TECHIP
优矽科技

目录

- RVV 0.8与A53 NEON指令集对比
- 程序测例比较
- 总结

UC TECHIP
优矽科技

RVV 0.8 vs A53 NEON

		RV[F/D] + RV-HP	基于FPUv4的扩展
Floating Point	单精度运算	YES	YES
	双精度运算	YES	YES
	半精度运算	YES（需要HP支持）	YES
	半精度与单精度之间转换	YES（需要HP支持）	YES
	单精度与双精度之间转换	YES	YES
	半精度与双精度之间转换	YES（需要HP支持）	YES
	支持Fused Multiply Accumulate(FMAC)	YES	YES
	规范化和非规范化数据均在硬件中处理	YES	YES
	Trapless Operation enabling fast execution	YES	YES
	动态浮点舍入模式	YES	YES
	HP/SP/DP浮点与32-bit整型之间的相互转换	YES（需要HP支持）	YES
	HP/SP/DP浮点与64-bit整型之间的相互转换	YES（需要HP支持）	YES
	HP/SP/DP浮点与32-bit定点之间的相互转换	NO	YES
	HP/SP/DP浮点与64-bit定点之间的相互转换	NO	YES
	VFP Vector操作	NO	NO

运算包括：加、减、乘、除、比较、乘加、开方（浮点）、符号注入（浮点）
FAMC：指的是VFMA/VFMS/VFNMA/VFNMS这些指令
HP：半精度浮点

UC TECHIP
优矽科技

RVV 0.8 vs A53 NEON

		RVV-v0.8.0+RV[F/D]+RV-HP	Cortex-A53 NEON
SIMD	vector-vector instructions	YES	YES
	vector-scalar instructions	YES（大部分指令支持）	绝大部分指令不支持操作数是通用标量寄存器
	vector-immediate instructions	YES（大部分指令支持）	绝大部分指令不支持操作数是立即数
	SIMD半精度浮点运算	YES（需要HP支持）	YES
	SIMD单精度浮点运算	YES	YES
	SIMD双精度浮点运算	YES	YES
	SIMD 8、16、32、64位有符号和无符号整型运算	YES	YES
	结构化数据加载功能	YES	YES
	寄存器组扩展	YES(一次最多可使用8个寄存器)	NO
	浮点指令的操作数可以来自整型通用寄存器	YES（需要Zfinx支持）	NO
	半精度/单精度/双精度之间的相互转换	YES（需要HP支持）	YES
	HP/SP/DP转同等位宽的16/32/64-bit整型	YES（需要HP支持）	YES
	16/32/64-bit整型转同等位宽的HP/SP/DP	YES（需要HP支持）	YES
	HP/SP/DP转同等位宽的16/32/64-bit定点	NO	不支持HP/16-bit定点组合
	16/32/64-bit定点转同等位宽的HP/SP/DP	NO	不支持HP/16-bit定点组合
	单位系数的8/16-bit多项式运算	NO	YES
	整型算术的两个源操作数可以相差一倍位宽	YES（部分指令支持）	YES（少部分指令支持）
	整型算术的目的操作数可以与源操作数相差一倍位宽	YES（部分指令支持）	YES（少部分指令支持）
	HP/SP/DP转不同等位宽的16/32/64-bit整型	YES	NO
	16/32/64-bit整型转不同等位宽的HP/SP/DP	YES	NO
	load fault-only-first	YES	NO
	原子操作指令	YES	NO
	整型归约、浮点归约	YES	YES
	Vector寄存器索引方式	v0-v31	32个128位寄存器V0-V31 32个64位寄存器D0-D31 32个32位位寄存器S0-S31 32个16位寄存器H0-H31 32个8位寄存器B0-B31
	一条指令的元素包含情况	VLEN=128-bit，8个寄存器为一组： 128个8-bit元素 64个16-bit元素 32个32-bit元素 16个64-bit元素 VLEN=128-bit，1个寄存器为一组： 16个8-bit元素 8个16-bit元素 4个32-bit元素 2个64-bit元素	16个8-bit元素 8个16-bit元素 4个32-bit元素 2个64-bit元素

RVV 0.8 vs A53 NEON

- 运行环境对比
  - RVV 0.8.0：使用spike指令模拟器，指令集选择RV64IMAFDCV，其中VARCH=vlen:128,elen:64,slen:128
  - A53 NEON: 四核A53处理器@1152MHz

- 选择的CMSIS DSP测例
  - arm_add_f32
  - arm_cmplx_mult_cmplx_f32
  - arm_mat_mult_f32

Case1: add_f32

```
void arm_add_f32(
  const float32_t * pSrcA,
  const float32_t * pSrcB,
        float32_t * pDst,
        uint32_t blockSize)
{
    uint32_t blkCnt;    /* Loop counter */

    float32x4_t vec1;
    float32x4_t vec2;
    float32x4_t res;

    /* Compute 4 outputs at a time */
    blkCnt = blockSize >> 2U;

    while (blkCnt > 0U)
    {
        vec1 = vld1q_f32(pSrcA);
        vec2 = vld1q_f32(pSrcB);
        res = vaddq_f32(vec1, vec2);
        vst1q_f32(pDst, res);
        pSrcA += 4;
        pSrcB += 4;
        pDst += 4;
        blkCnt--;
    }

    /* Tail */
    blkCnt = blockSize & 0x3;

    while (blkCnt > 0U)
    {
      *pDst++ = (*pSrcA++) + (*pSrcB++);
      blkCnt--;
    }
}
```

不需要处理 leftover

```
vsetvli t0, a0, e32, m8
vlw.v   v0, (a1)
vlw.v   v8, (a2)
vfadd.vv v16, v8, v0
vsw.v   v16, (a3)
sub     a0, a0, t0
slli    t0, t0, 2
add     a1, a1, t0
add     a2, a2, t0
add     a3, a3, t0
bnez    a0, 1b
ret
```

需要额外处理 leftover

```
ldr     q0, [x5],#16
subs    w4, w4, #0x1
ldr     q1, [x6],#16
fadd    v0.4s, v0.4s, v1.4s
str     q0, [x7],#16
b.ne    409570 <arm_add_f32+0x18>
sub     w4, w8, #0x1
add     x4, x4, #0x1
lsl     x4, x4, #4
add     x0, x0, x4
add     x1, x1, x4
add     x2, x2, x4
ands    w3, w3, #0x3
b.eq    4095d0 <arm_add_f32+0x78>
sub     w4, w3, #0x1
mov     x3, #0x0
add     x4, x4, #0x1
ldr     s0, [x0,x3,lsl #2]
ldr     s1, [x1,x3,lsl #2]
fadd    s0, s0, s1
str     s0, [x2,x3,lsl #2]
add     x3, x3, #0x1
cmp     x3, x4
b.ne    4095b4 <arm_add_f32+0x5c>
ret
```

Payload	NEON	RVV
16	46	133
32	70	133
64	118	144
128	214	166
256	406	210
512	790	298
1024	1,559	474
2048	3,561	827

指令数量对比

UC TECHIP 优矽科技

Case2: cmplx_mult_cmplx_f32

```
void arm_cmplx_mult_cmplx_f32(
  const float32_t * pSrcA,
  const float32_t * pSrcB,
        float32_t * pDst,
        uint32_t numSamples)
{
  uint32_t blkCnt;                               /* Loop counter */
  float32_t a, b, c, d;
  float32x4x2_t va, vb;
  float32x4_t real, imag;
  float32x4x2_t outCplx;

  /* Compute 4 outputs at a time */
  blkCnt = numSamples >> 2U;

  while (blkCnt > 0U)
  {
      va = vld2q_f32(pSrcA);
      vb = vld2q_f32(pSrcB);
      pSrcA += 8;
      pSrcB += 8;
      outCplx.val[0] = vmulq_f32(va.val[0], vb.val[0]);
      outCplx.val[0] = vmlsq_f32(outCplx.val[0], va.val[1], vb.val[1]);
      outCplx.val[1] = vmulq_f32(va.val[0], vb.val[1]);
      outCplx.val[1] = vmlaq_f32(outCplx.val[1], va.val[1], vb.val[0]);
      vst2q_f32(pDst, outCplx);
      pDst += 8;
      blkCnt--;
  }

  blkCnt = numSamples & 3;

  while (blkCnt > 0U)
  {
    a = *pSrcA++;
    b = *pSrcA++;
    c = *pSrcB++;
    d = *pSrcB++;
    *pDst++ = (a * c) - (b * d);
    *pDst++ = (a * d) + (b * c);
    blkCnt--;
  }
}
```

```
vsetvli    t0, a0, e32, m4
vmv.v.i    v28, 0
vlseg2e.v  v0, (a1)   # v0, v1, v2, v3    ->  nf = 0, Real
                      # v4, v5, v6, v7    ->  nf = 1, Img
vlseg2e.v  v8, (a2)   # v8,   v9, v10, v11   -> nf = 0, Real
                      # v12, v13, v14, v15   -> nf = 1, Img
vfmul.vv    v16, v0, v8
vfmul.vv    v20, v4, v12
vfsub.vv    v24, v16, v20 # v24, v25, v26, v27  -> result, Real

vfmul.vv    v16, v0, v12
vfmul.vv    v20, v4, v8
vfadd.vv    v28, v16, v20 # v28, v29, v30, v31 -> result, Img

vsseg2e.v  v24, (a3)

sub        a0, a0, t0
slli       t0, t0, 3  # every element has 4 bytes
add        a1, a1, t0
add        a2, a2, t0
add        a3, a3, t0
bnez       a0, 1b
ret
```

```
ld2     {v4.4s, v5.4s}, [x5], #32
subs    w4, w4, #0x1
ld2     {v2.4s, v3.4s}, [x6], #32
fneg    v16.4s, v5.4s
fmul    v7.4s, v4.4s, v2.4s
fmul    v6.4s, v5.4s, v2.4s
fmla    v7.4s, v16.4s, v3.4s
fmla    v6.4s, v4.4s, v3.4s
mov     v0.16b, v7.16b
mov     v1.16b, v6.16b
st2     {v0.4s, v1.4s}, [x7], #32
b.ne    4087e0 <arm_cmplx_mult_cmplx_f32+0x18>
sub     w4, w9, #0x1
add     x4, x4, #0x1
lsl     x4, x4, #5
add     x0, x0, x4
add     x1, x1, x4
add     x2, x2, x4
ands    w3, w3, #0x3
b.eq    408860 <arm_cmplx_mult_cmplx_f32+0x98>
ldr     s0, [x0],#8
subs    w3, w3, #0x1
ldr     s1, [x1],#8
ldur    s2, [x0,#-4]
ldur    s3, [x1,#-4]
fmul    s4, s2, s1
fmul    s2, s2, s3
fnmsub  s1, s0, s1, s2
fmadd   s0, s0, s3, s4
str     s1, [x2],#8
stur    s0, [x2,#-4]
b.ne    408830 <arm_cmplx_mult_cmplx_f32+0x68>
ret
```

Payload	NEON	RVV
16	70	139
32	118	156
64	214	190
128	406	258
256	790	394
512	1,560	666
1024	3,097	1,210

UC TECHIP
优矽科技

Case3:mat_mult_f32

```
/* Matrix multiplication */
while (colCnt > 0U)
{
  /* c(m,n) = a(1,1)*b(1,1) + a(1,2)*b(2,1) + ... + a(m,p)*b(p,n) */
  a0V = vld1q_f32(pIn1);
  a1V = vld1q_f32(pIn1B);
  a2V = vld1q_f32(pIn1C);
  a3V = vld1q_f32(pIn1D);
  a4V = vld1q_f32(pIn1E);
  a5V = vld1q_f32(pIn1F);
  a6V = vld1q_f32(pIn1G);
  a7V = vld1q_f32(pIn1H);

  pIn1 += 4;
  pIn1B += 4;
  pIn1C += 4;
  pIn1D += 4;
  pIn1E += 4;
  pIn1F += 4;
  pIn1G += 4;
  pIn1H += 4;

  temp[0] = *pIn2;
  pIn2 += numColsB;
  temp[1] = *pIn2;
  pIn2 += numColsB;
  temp[2] = *pIn2;
  pIn2 += numColsB;
  temp[3] = *pIn2;
  pIn2 += numColsB;

  acc0 = vmlaq_f32(acc0,a0V,temp);
  acc1 = vmlaq_f32(acc1,a1V,temp);
  acc2 = vmlaq_f32(acc2,a2V,temp);
  acc3 = vmlaq_f32(acc3,a3V,temp);
  acc4 = vmlaq_f32(acc4,a4V,temp);
  acc5 = vmlaq_f32(acc5,a5V,temp);
  acc6 = vmlaq_f32(acc6,a6V,temp);
  acc7 = vmlaq_f32(acc7,a7V,temp);

  /* Decrement the loop count */
  colCnt--;
}
```

```
vsetvli t0, a0, e32, m8      # a0 = colCnt
vlsw.v v0, (a2), a1          # a1 = numColsB*4, a2 = pIn2
vlw.v  v8, (a3)              # a3 = pIn1
vfmul.vv    v16, v0,  v8
vfredsum.vs v24, v16, v24
vfmv.f.s    ft1,  v24
vmv.v.i     v24, 0
fadd.s      fa5, fa5, ft1
sub         a0, a0, t0
slli        t0, t0, 2        # t0 to bytes
add         a3, a3, t0       # pIn1 += t0
mul         t1, t0, a6       # t1 = numColsB*t0
add         a2, a2, t1       # pIn2 += numColsB*t0
bnez        a0, 1b
fsw         fa5, (a4)
ret
```

payload	NEON	RVV
4x4	634	797
8x8	1414	2,661
16x16	6,804	10,037
32x32	40,284	39,381

总结

- 可变的向量长度
- 同样的程序可以适应不同的向量处理器架构
- 向量指令支持掩码操作

UC TECHIP
优矽科技

Thank You!