x264 RISC-V生态构建和优化

钱佳炎
qianjiayan.1@bytedance.com
字节跳动服务器团队

ByteDance 字节跳动

目录

1. 背景

2. x264生态构建

3. RVV指令集瓶颈和优化

4. 从x264看RISC-V软件生态的挑战

背景

视频编解码生态现状：

- 对RISC-V的支持刚刚起步，且集中在h264，dav1d等解码器领域
- 以FFmpeg的架构相关文件数量为例，生态差距较大，但也在快速发展：

架构	文件数量 in FFmpeg	百分比
RISC-V	61	8.5%
ARM/NEON	293	40.7%
x86/AVX/SSE	365	50.8%

我们的工作：

- 团队致力于推动面向字节业务的RISCV软件生态快速发展
- 视频编解码是字节数据中心的重要场景之一，其中x264/x265编码器是优化重点

目录

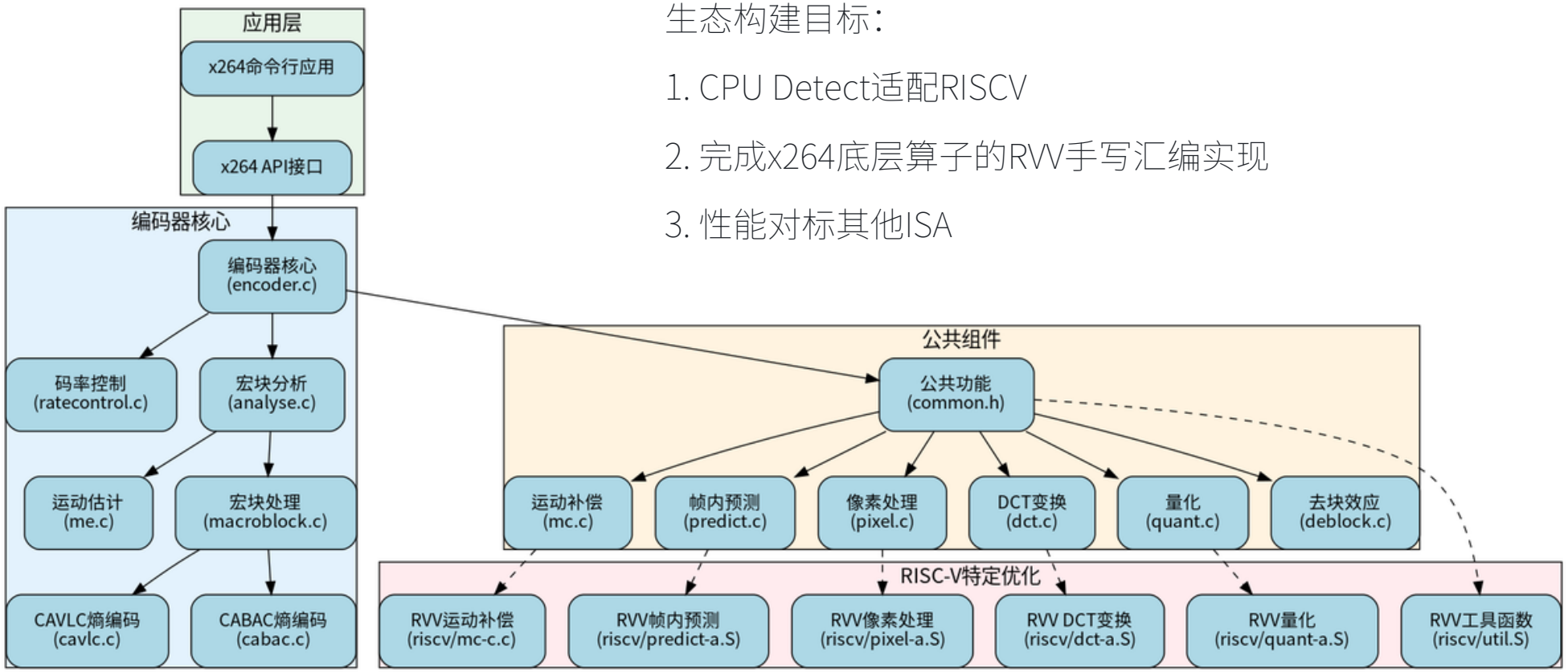
1. 背景

2. x264生态构建

3. RVV指令集瓶颈和优化

4. 从x264看RISC-V软件生态的挑战

x264生态构建



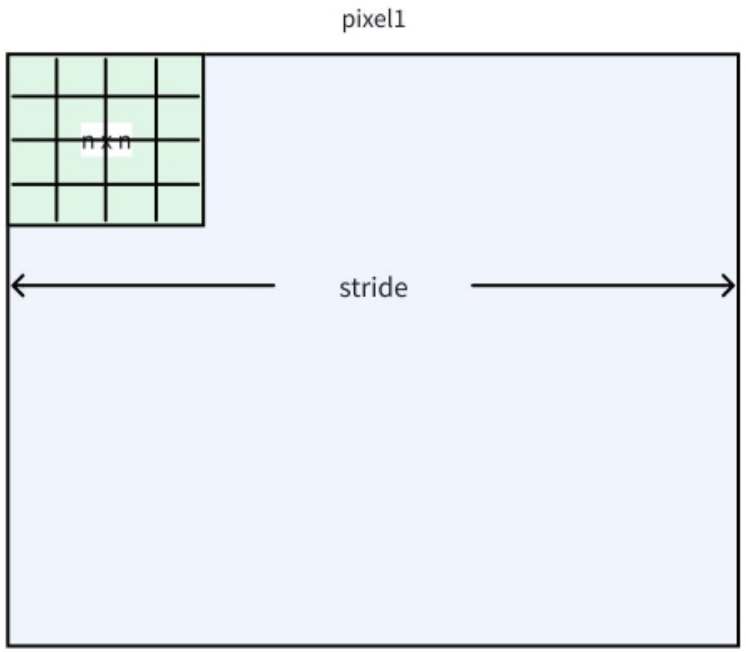
生态构建目标：

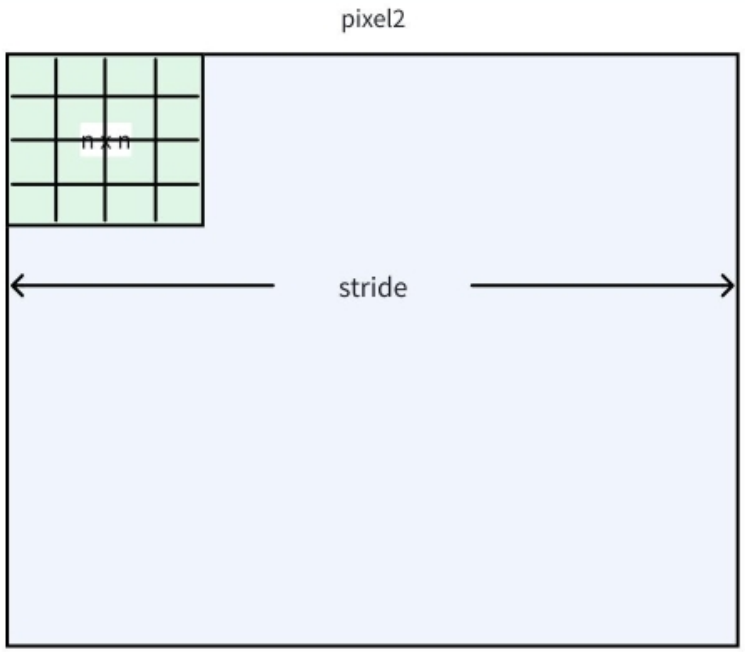
1. CPU Detect适配RISCV
2. 完成x264底层算子的RVV手写汇编实现
3. 性能对标其他ISA

x264生态构建

x264典型计算pattern：

- 在两帧图像中各取一个固定大小的像素块（4x4 ~ 16x16），做相关运算
- 在传统x86/arm CPU上，x264更适配高并发的窄向量





x264生态构建

x264典型RVV实现案例：

- 利用stride load填满向量寄存器
  - 通过SEW=32/64读一行的多个u8
  - 依赖硬件支持非对齐访问
- register group配合widening计算显著的减少前端指令数
- VLA实现，支持任意vlen>=128

潜在问题：

- 访存-计算无法交替执行隐藏时延
- 对大vlen硬件性能不一定最优
  - 除非指令性能依赖vl而不是依赖LMUL

rvv — Assembly language

```
1   function pixel_ssd_4x8_rvv
2   vsetivli    zero, 8, e32, m2, ta, ma
3   vlse32.v    v0, (a0), a1
4   vlse32.v    v2, (a2), a3
5   li          t0, 32
6   vsetvli     zero, t0, e8, m2, ta, ma
7   vwsubu.vv   v4, v0, v2 //输出v4~v7
8   vsetvili    zero, 16, e16, m2, ta, ma
9   vwmul.vv    v16, v4, v4
10  vwmacc.vv   v16, v6, v6 //输出v16~v19
11  vsetivli    zero, 16, e32, m4, ta, ma
12  vmv.s.x     v31, zero
13  vredsum.vs  v31, v16, v31
14  vmv.x.s     a0, v31
15
16  ret
17  endfunc
```

neon

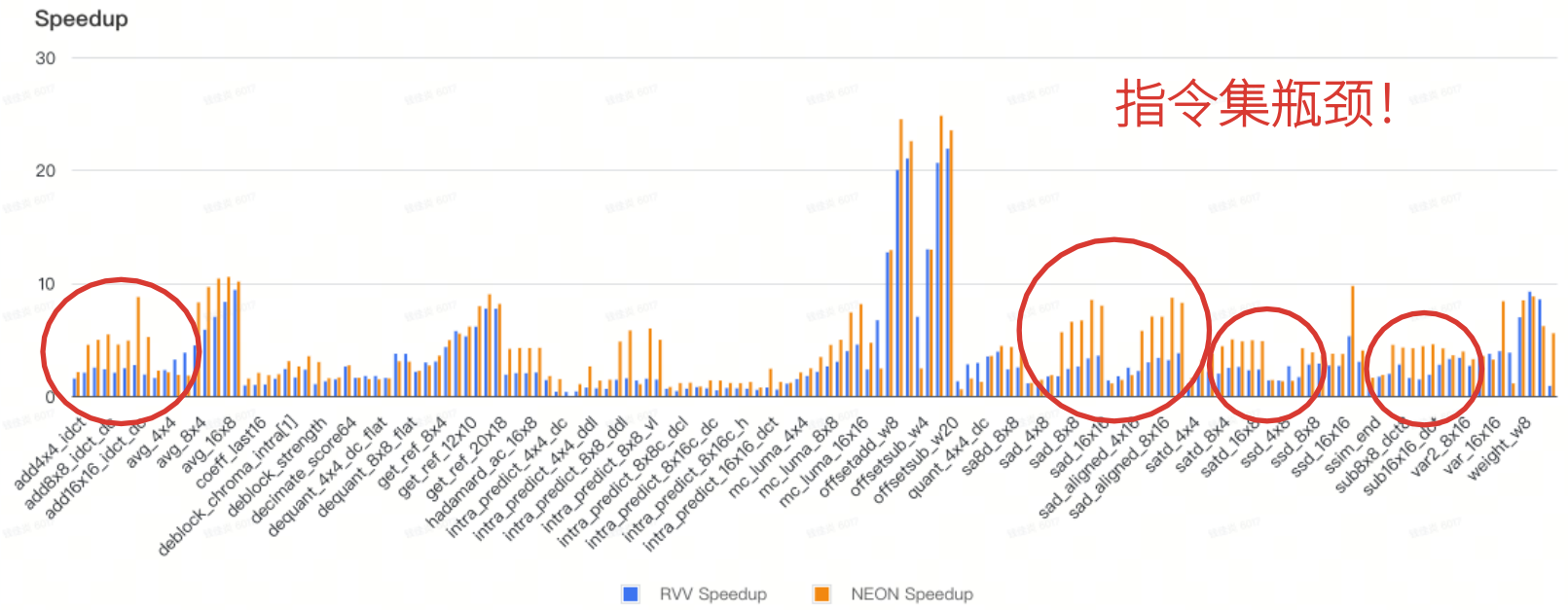
```
1   function pixel_ssd_4x8_neon
2   ld1         {v16.s}[0], [x0], x1
3   ld1         {v17.s}[0], [x2], x3
4   usubl         v2.8h, v16.8b, v17.8b
5   ld1         {v16.s}[0], [x0], x1
6   ld1         {v17.s}[0], [x2], x3
7   smull         v0.4s, v2.4h, v2.4h
8   usubl         v2.8h, v16.8b, v17.8b
9   ld1         {v16.s}[0], [x0], x1
10  ld1         {v17.s}[0], [x2], x3
11  smlal         v0.4s, v2.4h, v2.4h
12  usubl         v2.8h, v16.8b, v17.8b
13  ld1         {v16.s}[0], [x0], x1
14  ld1         {v17.s}[0], [x2], x3
15  smlal         v0.4s, v2.4h, v2.4h
16  usubl         v2.8h, v16.8b, v17.8b
17  ld1         {v16.s}[0], [x0], x1
18  ld1         {v17.s}[0], [x2], x3
19  smlal         v0.4s, v2.4h, v2.4h
20  usubl         v2.8h, v16.8b, v17.8b
21  ld1         {v16.s}[0], [x0], x1
22  ld1         {v17.s}[0], [x2], x3
23  smlal         v0.4s, v2.4h, v2.4h
24  usubl         v2.8h, v16.8b, v17.8b
25  ld1         {v16.s}[0], [x0], x1
26  ld1         {v17.s}[0], [x2], x3
27  smlal         v0.4s, v2.4h, v2.4h
28  usubl         v2.8h, v16.8b, v17.8b
29  ld1         {v16.s}[0], [x0], x1
30  ld1         {v17.s}[0], [x2], x3
31  smlal         v0.4s, v2.4h, v2.4h
32  usubl         v2.8h, v16.8b, v17.8b
33  smlal         v0.4s, v2.4h, v2.4h
34  addv         s0, v0.4s
35  mov         w0, v0.s[0]
36  ret
```

x264生态构建

由于缺乏高性能RISC-V CPU，
采用**相对**的向量加速比替代**绝对**性能来评估实现效果

测试平台：

- SpaceMIT K1
  - vlen=256 dlen=128
- ARM N2
  - 2x128 neon



指令集瓶颈！

https://code.videolan.org/BD-qjy/x264-riscv-dev

目录

1. 背景

2. x264生态构建

3. RVV指令集瓶颈和优化

4. 从x264看RISC-V软件生态的挑战

RVV指令集瓶颈

1. in-register transpose
2. absolute difference
2. Signed saturate and Narrow to Unsigned
3. zero-extend move from scalar to vector

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	5

NEON：

- trn1/trn2指令组合，nlogn的复杂度

RVV：

- segment load/store
- LMUL=n的vrgather
- vnsrl模拟zip
- else

RVV指令集瓶颈

1. in-register transpose
2. absolute difference
2. Signed saturate and Narrow to Unsigned
3. zero-extend move from scalar to vector

C = abs(A – B)

```
.macro uabd d0, s0, s1, t0
  vmaxu.vv      \d0, \s0, \s1
  vminu.vv      \t0, \s0, \s1
  vsub.vv       \d0, \d0, \t0
.endm
```

C += abs(A – B)

```
.macro uabal d0, s0, s1, t0, t1
  vmaxu.vv      \t1, \s0, \s1
  vminu.vv      \t0, \s0, \s1
  vsub.vv       \t0, \t1, \t0
  vwaddu.wv     \d0, \d0, \t0
.endm
```

RVV指令集瓶颈

1. in-register transpose
2. absolute difference
3. Signed saturate and Narrow to Unsigned
4. zero-extend move from scalar to vector

For `vnclipu`, the shifted rounded source value is treated as an unsigned integer and saturates if the result would overflow the destination viewed as an unsigned integer.

> **Note** There is no single instruction that can saturate a signed value into an unsigned destination. A sequence of two vector instructions that first removes negative numbers by performing a max against 0 using `vmax` then clips the resulting unsigned value into the destination using `vnclipu` can be used if setting `vxsat` value for negative numbers is not required. A `vsetvli` is required inbetween these two instructions to change SEW.

For `vnclip`, the shifted rounded source value is treated as a signed integer and saturates if the result would overflow the destination viewed as a signed integer.

```
vsetvli    zero, zero, e16, m2, ta, ma
vmax.vx    v2, v2, zero
vsetvli    zero, zero, e8, m1, ta, ma
vnclipu.wi  v8, v2, 0
```

RVV指令集瓶颈

1. in-register transpose
2. absolute difference
2. Signed saturate and Narrow to Unsigned
3. zero-extend move from scalar to vector

The `vmv.x.s` instruction copies a single SEW-wide element from index 0 of the source vector register to a destination integer register. If SEW > XLEN, the least-significant XLEN bits are transferred and the upper SEW-XLEN bits are ignored. If SEW < XLEN, the value is sign-extended to XLEN bits.

```
vsetvli zero, zero, e16, m1, ta, ma
vmv.x.s a0, v1
zext.h a0, a0
```

推动指令集优化

引起共识

1. 在RISE提出问题，收集更多生态开发者的反馈
2. 在Vector Sig详细讨论Gap，形成Sig的25年重点工作之一

讨论提案

1. Zvabd from ByteDance
--优化 absolute difference
2. Zvzip from Rivos
--优化 in-register transpose

正式推进

1. Fast Track of Zvabd (integer vector absolute difference) in progress

推动指令集优化（Zvabd from ByteDance）

- 当前提议的指令功能如下：

```
# Signed Integer Absolute
vabs.v    vd, vs2, vm   # vd[i] = abs(vs2[i])
# Signed Integer Absolute Difference
vabd.vv    vd, vs2, vs1, vm   # vd[i] = abs(vs2[i] - vs1[i])
# Unsigned Integer Absolute Difference
vabdu.vv   vd, vs2, vs1, vm   # vd[i] = abs(vs2[i] - vs1[i])
# Widening signed absolute difference accumulate, overwrite addend
vwabdacc.vv vd, vs2, vs1, vm # vd[i] = abs(vs2[i] - vs1[i]) + vd[i]
# Widening unsigned absolute difference accumulate, overwrite addend
vwabdaccu.vv vd, vs2, vs1, vm # vd[i] = abs(vs2[i] - vs1[i]) + vd[i]
```

- 已经正式进入Fast Track流程
- 社区对新指令扩展的要求：
  - 价值明确
    - 应用级的收益
  - 功能最小化
    - opcode
    - .vi/.vx/.vv
    - SEW

目录

1. 背景

2. x264生态构建

3. RVV指令集瓶颈和优化

4. 从x264看RISC-V软件生态的挑战

从x264看RISC-V软件生态的挑战

- 指令集仍然有一定的差距，但在快速补齐中
- vector len的碎片化
  - 向量设计的分化：
    - 顺序宽向量
    - 乱序窄向量
  - VLA的理想和现实：
    - 理想：一套代码适配任意vlen
    - 现实：每个特定的vlen都需要独立的最优实现
- 缺失高性能开发验证平台，软件实现当前以porting为主，还有很多优化空间

THANKS

ByteDance字节跳动