中国科学院计算技术研究所 INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

北京开源芯片研究院 BEIJING INSTITUTE OF OPEN SOURCE CHIP

香山开源处理器社区 XiangShan Open-Source Processor Community



面向通用高性能计算的向量扩展设计

胡轩[1] 王桑平[4] 赵祖康[4] 程光辉[2] 游朝阳[2] 胡格致[3] 肖飞豹[2]

吉骏雄[1] 朱夏楠[1] 白冰洋[5] 蔡驰[6] 张紫飞[2]

[1]中国科学院计算技术研究所 [2]北京开源芯片研究院

[3]中国科学院大学 [4]上海科技大学 [5]深圳技术大学 [6]北京邮电大学

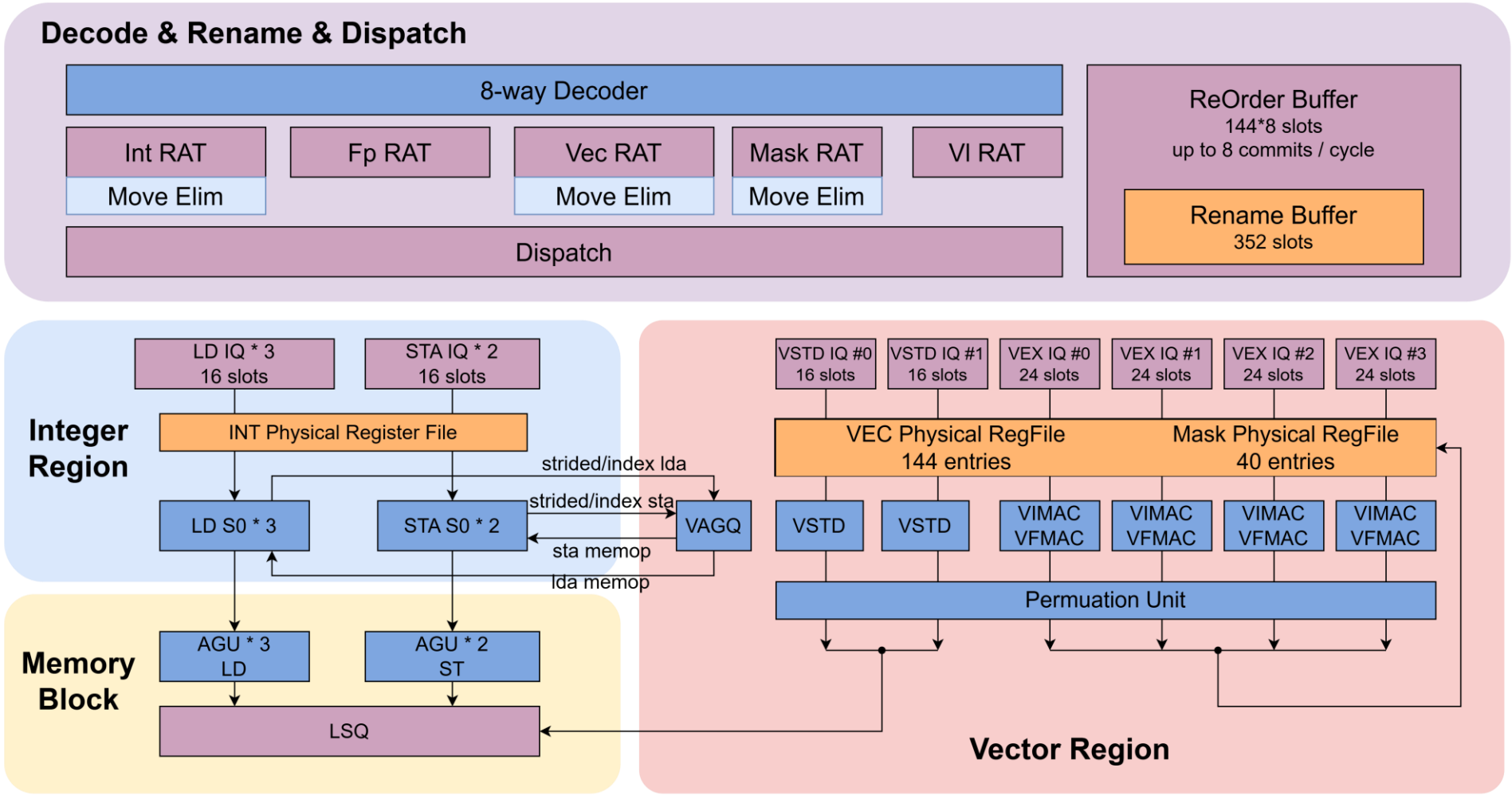
2026 年 10 月 20 日@第五届 RISC-V 中国峰会

RVV 的通算能力备受质疑

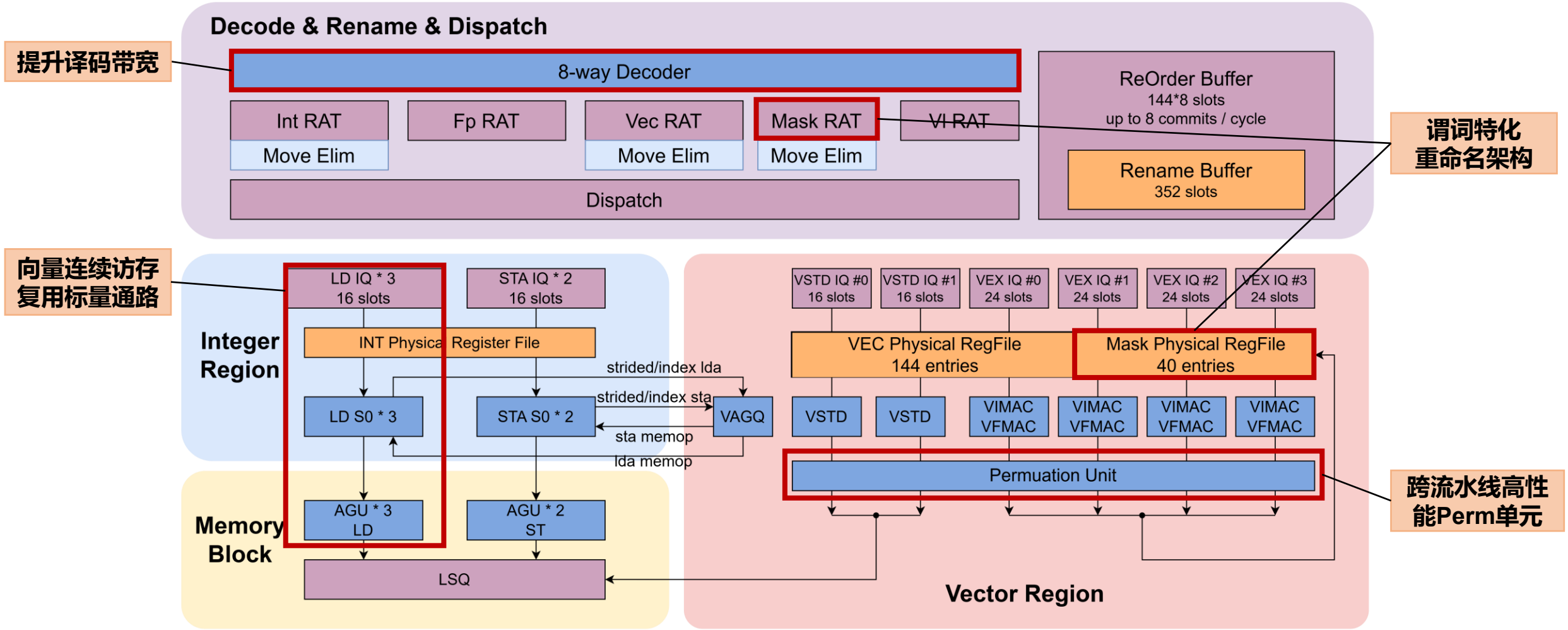
- RVV 规范冻结至今已有 5 年
- 众多“特性”饱受诟病
  - 变长向量
  - 独立向量配置指令
  - 没有独立的谓词寄存器
  - LMUL 机制
  - LMUL>1 时的 vrgather 等 Perm 指令
  - Segment 访存
  - ……
- RVV 似乎不是一个为通用 CPU 设计的指令集？
- **经过两年的研发，香山新向量架构聚焦解决以上问题**



香山昆明湖 V3 向量架构总览

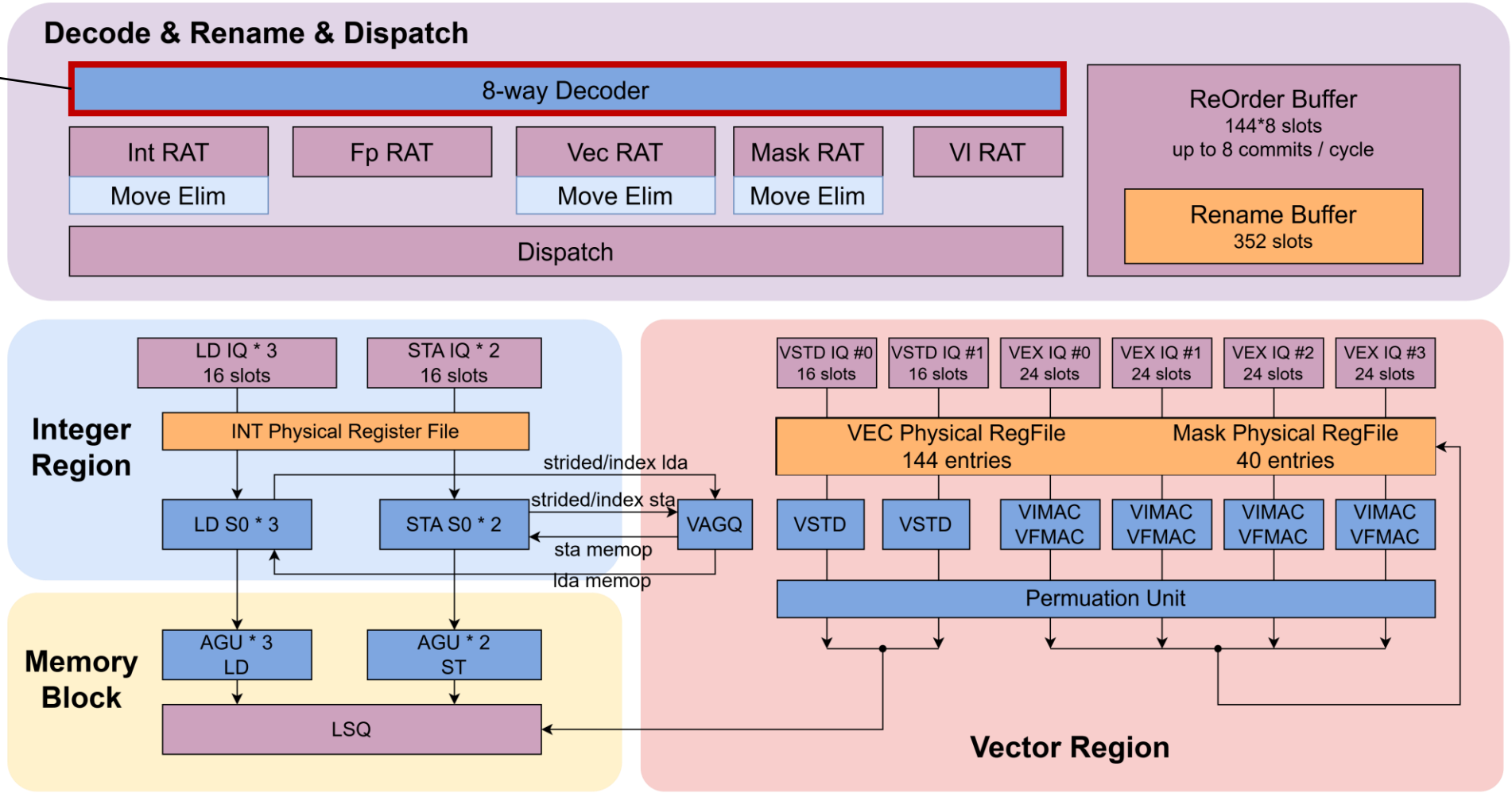


向量架构重要优化总览


提升译码带宽
Decode & Rename & Dispatch
8-way Decoder
Int RAT
Move Elim
Fp RAT
Vec RAT
Move Elim
Mask RAT
Move Elim
VI RAT
Dispatch
ReOrder Buffer
144*8 slots
up to 8 commits / cycle
Rename Buffer
352 slots
谓词特化
重命名架构
向量连续访存
复用标量通路
Integer
Region
LD IQ * 3
16 slots
STA IQ * 2
16 slots
INT Physical Register File
strided/index lda
strided/index sta
LD S0 * 3
STA S0 * 2
VAGQ
sta memop
lda memop
Memory
Block
AGU * 3
LD
AGU * 2
ST
LSQ
VSTD IQ #0
16 slots
VSTD IQ #1
16 slots
VEX IQ #0
24 slots
VEX IQ #1
24 slots
VEX IQ #2
24 slots
VEX IQ #3
24 slots
VEC Physical RegFile
144 entries
Mask Physical RegFile
40 entries
VSTD
VSTD
VIMAC
VFMAC
VIMAC
VFMAC
VIMAC
VFMAC
VIMAC
VFMAC
Permuation Unit
跨流水线高性
能Perm单元
Vector Region

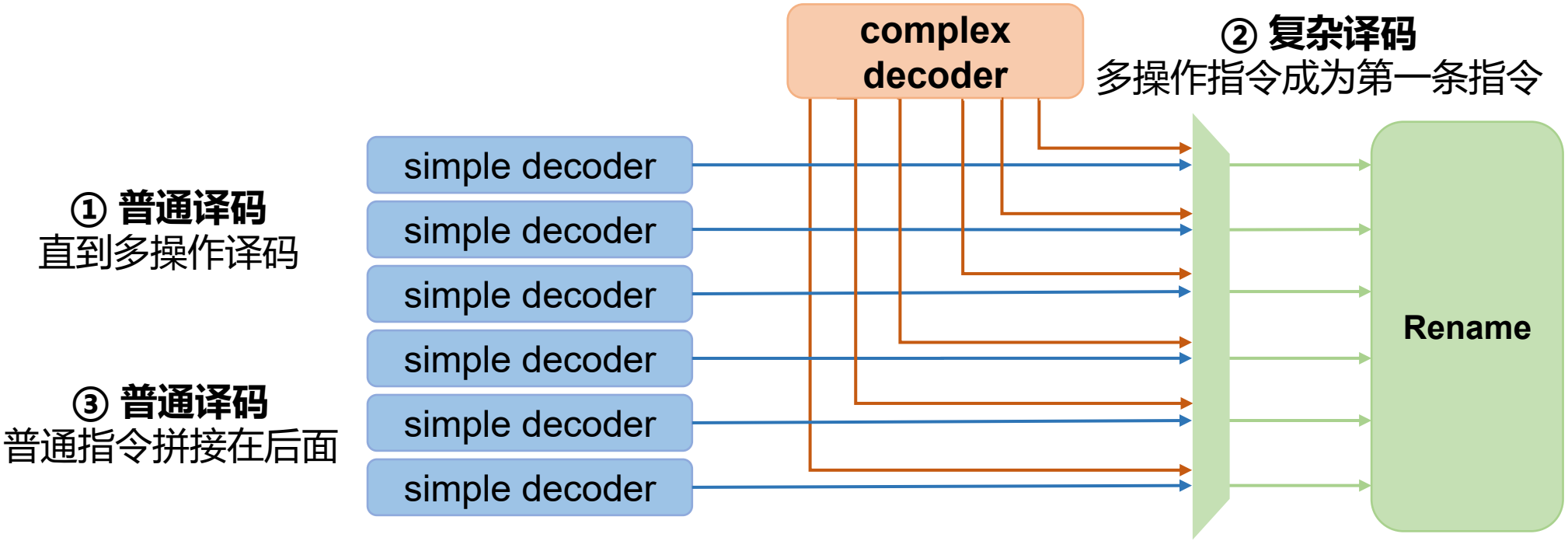
架构优化——译码

提升译码带宽



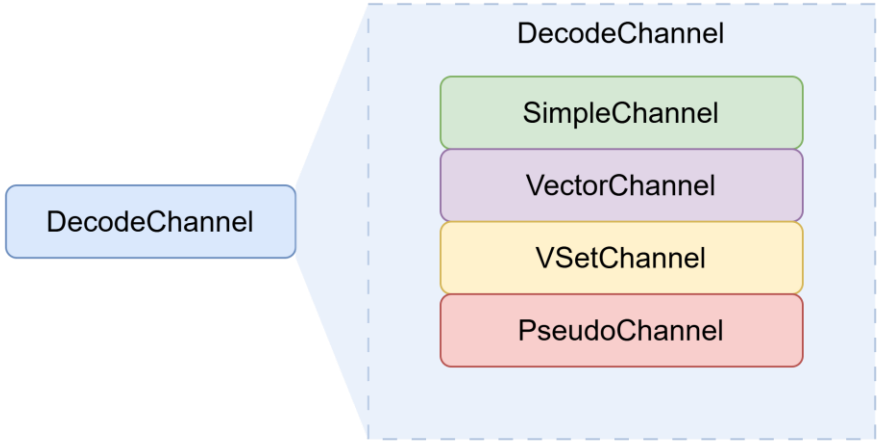
旧版译码实现成为向量瓶颈

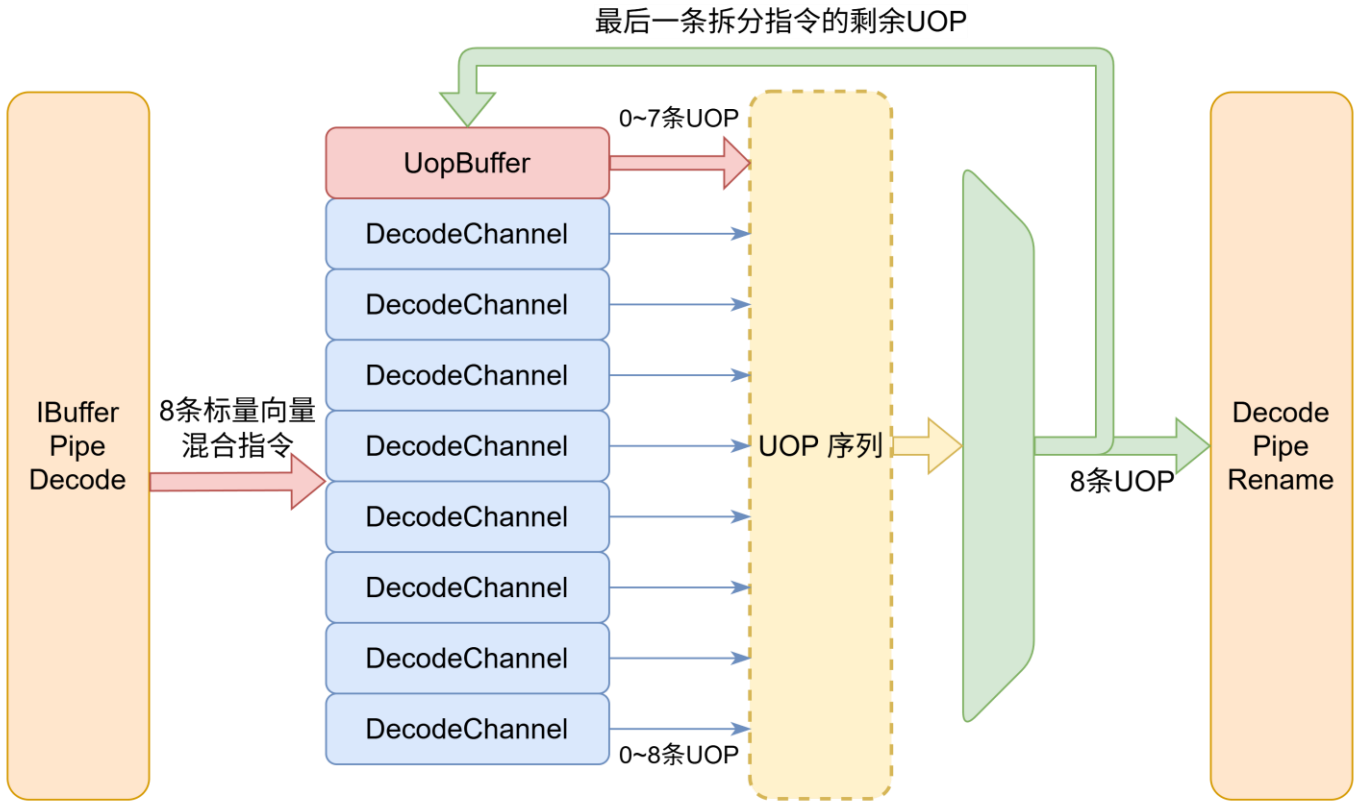
- Simple Decoder：普通指令译码器，每条译码通道各一个
- Complex Decoder：多操作指令译码器，多个译码通道共享一个
- 每周期只能输出一条LMUL>1的向量指令UOP
- 单指令拆分多达 64 条 UOP，严重阻塞译码



新版译码无空泡供应 UOP

- **任意混合指令流无空泡 UOP 输出**
  - 除seg3/5/6/7 load/store指令
- 多达 8 个可拆分指令通道
- 每个通道由四个子通道组成
  - Simple　　简单指令
  - Vector　　向量和AMOCAS拆分
  - VSet　　向量配置指令
  - Pseudo　　CSRR VL、软件预取等
- 每通道输出 0/1/2/4/8 条UOP
- UOP 序列前 8 条 UOP 进入重命名
- UopBuffer 暂存已译码未送出的 UOP





一个向量加法例子

```c
#include <riscv_vector.h>
#include <stdint.h>

void vadd(
  uint32_t* __restrict vd,
  uint32_t* __restrict vs1,
  uint32_t* __restrict vs2,
  size_t n
) {
  for (size_t i = 0; i < n; i++) {
    vd[i] = vs1[i] + vs2[i];
  }
}
```

```
vadd:
        beq     a3,zero,.L11
.L3:
        vsetvli a5,a3,e32,m4,ta,ma   1
        vle32.v v8,0(a1)             4
        vle32.v v4,0(a2)             4
        sub     a3,a3,a5             1
        sh2add  a1,a5,a1             1
        sh2add  a2,a5,a2             1
        vadd.vv v0,v4,v8             4
        vse32.v v0,0(a0)             4
        sh2add  a0,a5,a0             1
        bne     a3,zero,.L3          1
.L11:
        ret
```

LMUL=4

UOP 数量

向量加法举例*

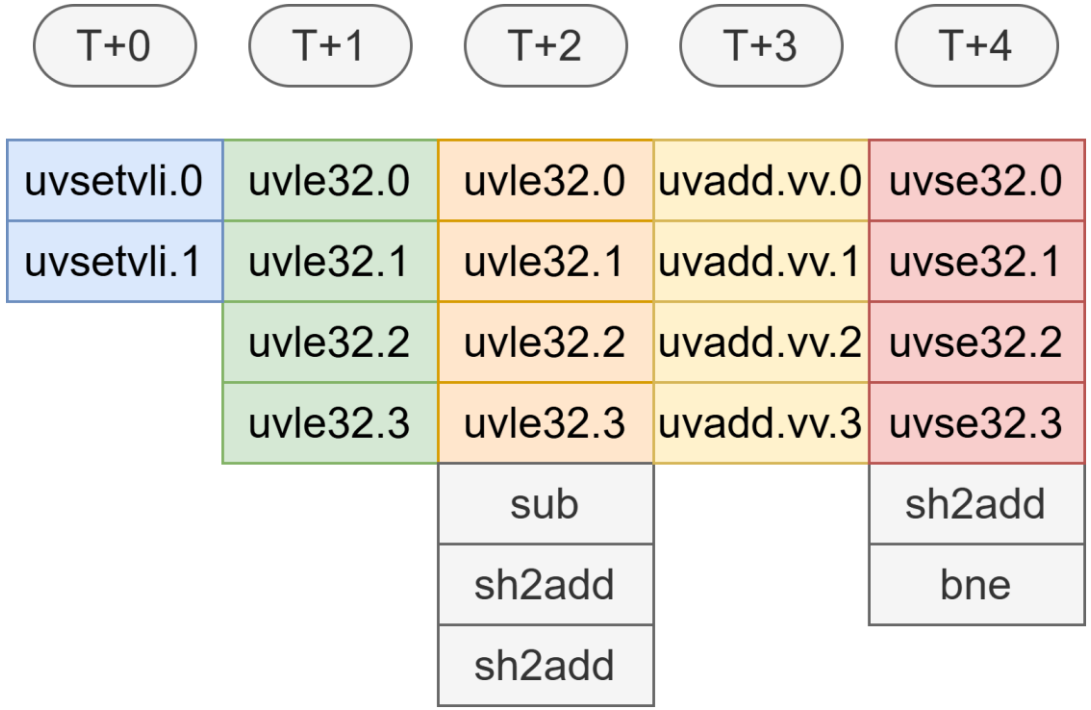
https://godbolt.org/z/6GoWWP6nh

举例：旧译码产生空泡

- kmhv2 仅有一个拆分单元
- 支持非拆分指令尾填充

```
1   vadd:
2           beq     a3,zero,.L11          UOP 数量
3   .L3:
4           vsetvli a5,a3,e32,m4,ta,ma    1
5           vle32.v v8,0(a1)              4
6           vle32.v v4,0(a2)              4
7           sub     a3,a3,a5              1
8           sh2add  a1,a5,a1              1
9           sh2add  a2,a5,a2              1
10          vadd.vv v0,v4,v8              4
11          vse32.v v0,0(a0)              4
12          sh2add  a0,a5,a0              1
13          bne     a3,zero,.L3           1
14  .L11:
15          ret
```

向量加法汇编 LMUL=4



kmhv2 译码时空图

举例：新译码无空泡

- kmhv3 译码可配置至多 8 个拆分单元
- 无空泡拆分标量向量混合指令流

```
1   vadd:
2           beq     a3,zero,.L11        UOP 数量
3   .L3:
4           vsetvli a5,a3,e32,m4,ta,ma  1
5           vle32.v v8,0(a1)            4
6           vle32.v v4,0(a2)            4
7           sub     a3,a3,a5            1
8           sh2add  a1,a5,a1            1
9           sh2add  a2,a5,a2            1
10          vadd.vv v0,v4,v8            4
11          vse32.v v0,0(a0)            4
12          sh2add  a0,a5,a0            1
13          bne     a3,zero,.L3         1
14  .L11:
15          ret
```

向量加法汇编 LMUL=4

小优化：kmhv3 支持单 uop 双写 int vl



uvsetvli	uvle32.3	uvse32.0	uvle32.1	sh2add
uvle32.0	sub	uvse32.1	uvle32.2	sh2add
uvle32.1	sh2add	uvse32.2	uvle32.3	uvadd.vv.0
uvle32.2	sh2add	uvse32.3	uvle32.0	uvadd.vv.1
uvle32.3	uvadd.vv.0	sh2add	uvle32.1	uvadd.vv.2
uvle32.0	uvadd.vv.1	bne	uvle32.2	uvadd.vv.3
uvle32.1	uvadd.vv.2	uvsetvli	uvle32.3	uvse32.0
uvle32.2	uvadd.vv.3	uvle32.0	sub	uvse32.1

kmhv3 译码时空图

译码优化前后对比

- 优化后显著提升标量-向量混合 UOP 吞吐

T+0	T+1	T+2	T+3	T+4
uvsetvli.0	uvle32.0	uvle32.0	uvadd.vv.0	uvse32.0
uvsetvli.1	uvle32.1	uvle32.1	uvadd.vv.1	uvse32.1
	uvle32.2	uvle32.2	uvadd.vv.2	uvse32.2
	uvle32.3	uvle32.3	uvadd.vv.3	uvse32.3
		sub		sh2add
		sh2add		bne
		sh2add		

kmhv2 译码时空图

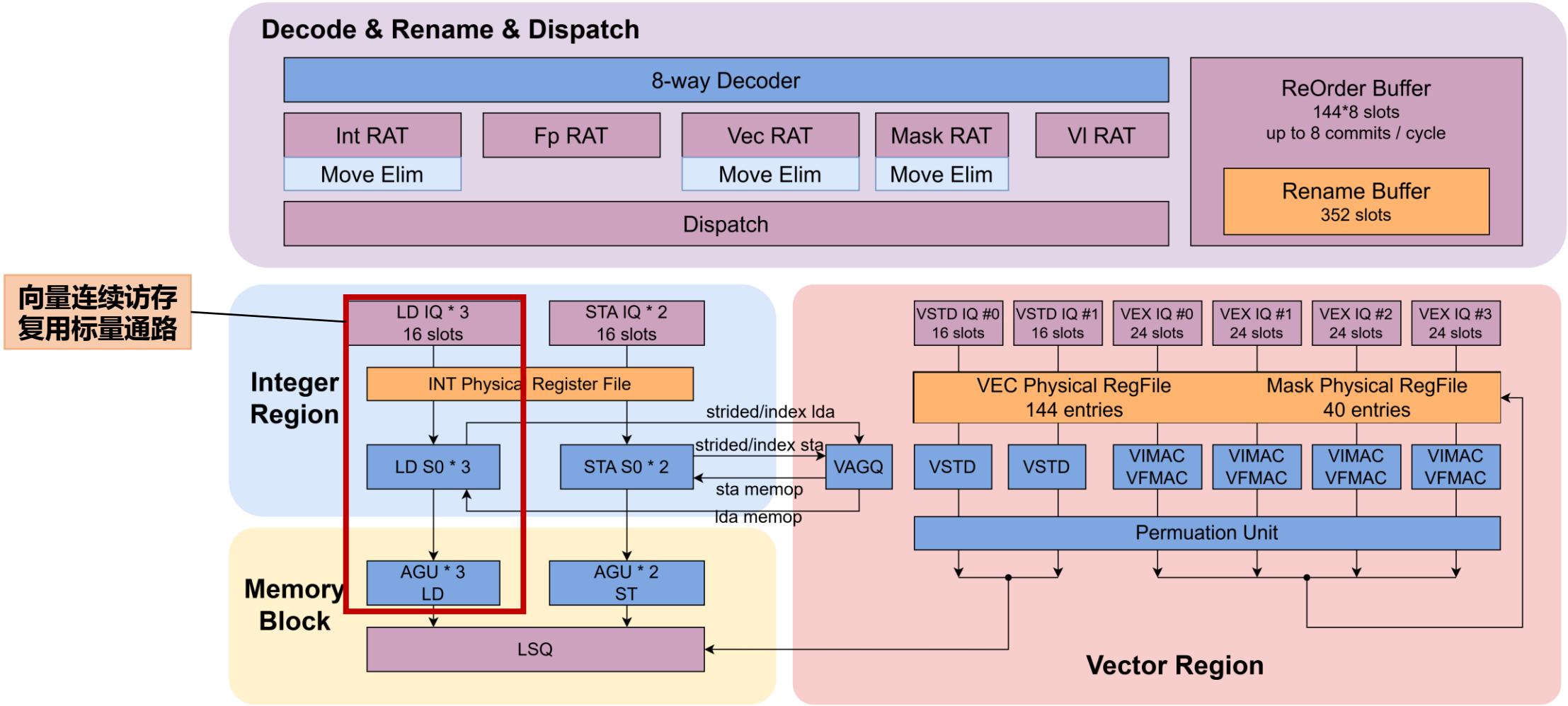
小优化：kmhv3 支持单 uop 双写 int vl



T+0	T+1	T+2	T+3	T+4
uvsetvli	uvle32.3	uvse32.0	uvle32.1	sh2add
uvle32.0	sub	uvse32.1	uvle32.2	sh2add
uvle32.1	sh2add	uvse32.2	uvle32.3	uvadd.vv.0
uvle32.2	sh2add	uvse32.3	uvle32.0	uvadd.vv.1
uvle32.3	uvadd.vv.0	sh2add	uvle32.1	uvadd.vv.2
uvle32.0	uvadd.vv.1	bne	uvle32.2	uvadd.vv.3
uvle32.1	uvadd.vv.2	uvsetvli	uvle32.3	uvse32.0
uvle32.2	uvadd.vv.3	uvle32.0	sub	uvse32.1

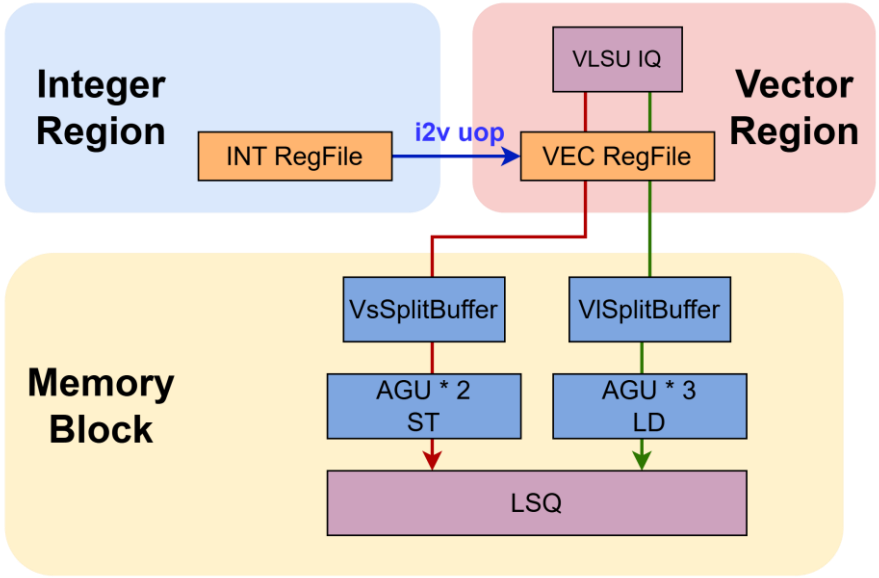
kmhv3 译码时空图

架构优化——向量连续访存

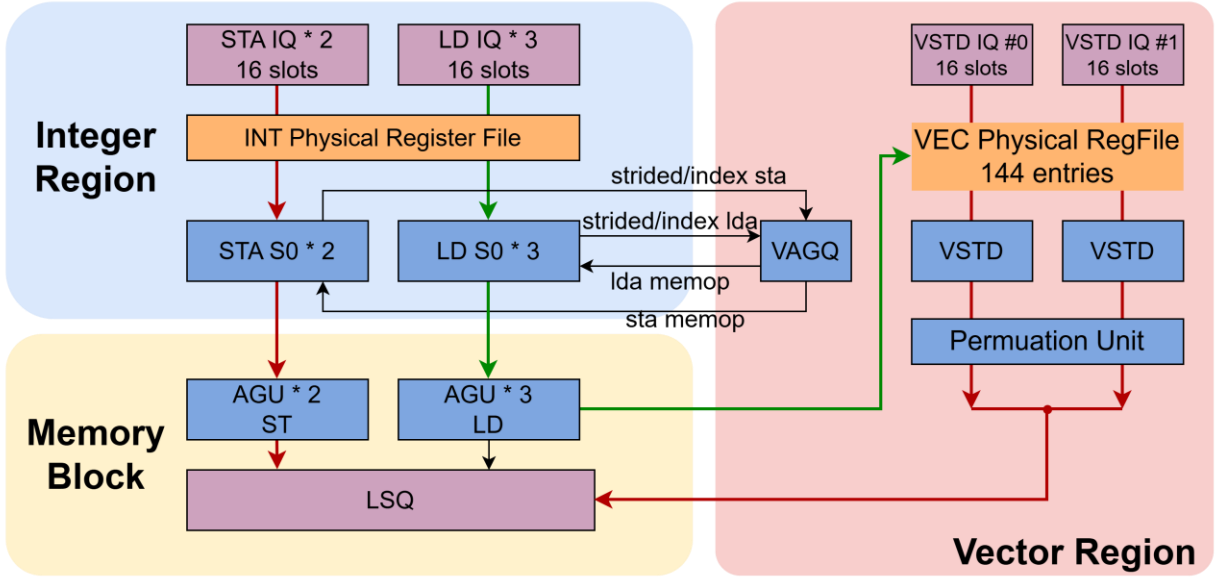

Decode & Rename & Dispatch
8-way Decoder
Int RAT
Move Elim
Fp RAT
Vec RAT
Move Elim
Mask RAT
Move Elim
VI RAT
Dispatch
ReOrder Buffer
144*8 slots
up to 8 commits / cycle
Rename Buffer
352 slots
向量连续访存
复用标量通路
Integer Region
LD IQ * 3
16 slots
STA IQ * 2
16 slots
INT Physical Register File
LD S0 * 3
STA S0 * 2
VAGQ
strided/index lda
strided/index sta
sta memop
lda memop
Memory Block
AGU * 3
LD
AGU * 2
ST
LSQ
VSTD IQ #0
16 slots
VSTD IQ #1
16 slots
VEX IQ #0
24 slots
VEX IQ #1
24 slots
VEX IQ #2
24 slots
VEX IQ #3
24 slots
VEC Physical RegFile
144 entries
Mask Physical RegFile
40 entries
VSTD
VSTD
VIMAC
VFMAC
VIMAC
VFMAC
VIMAC
VFMAC
VIMAC
VFMAC
Permuation Unit
Vector Region

向量连续访存优化

- kmhv2 vle/vse 指令延迟高，吞吐低
  - 额外UOP将地址操作数搬移到向量域
  - SplitBuffer 内对齐拆分，加深流水级
  - SplitBuffer 容量限制吞吐和乱序窗口



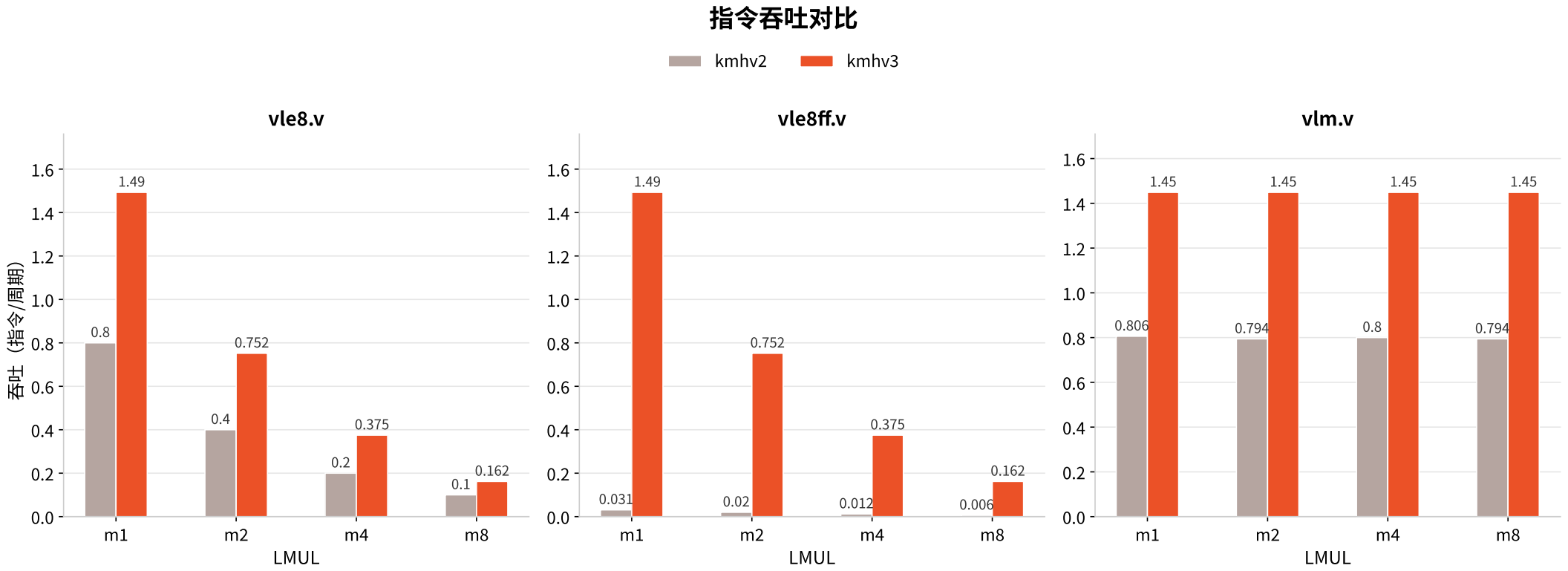
- kmhv3 新向量
  - **复用标量访存发射通路**
  - 向量连续访存无 buffer
  - 利用 v3 访存高效非对齐实现
  - 吞吐和乱序窗口与标量访存相等



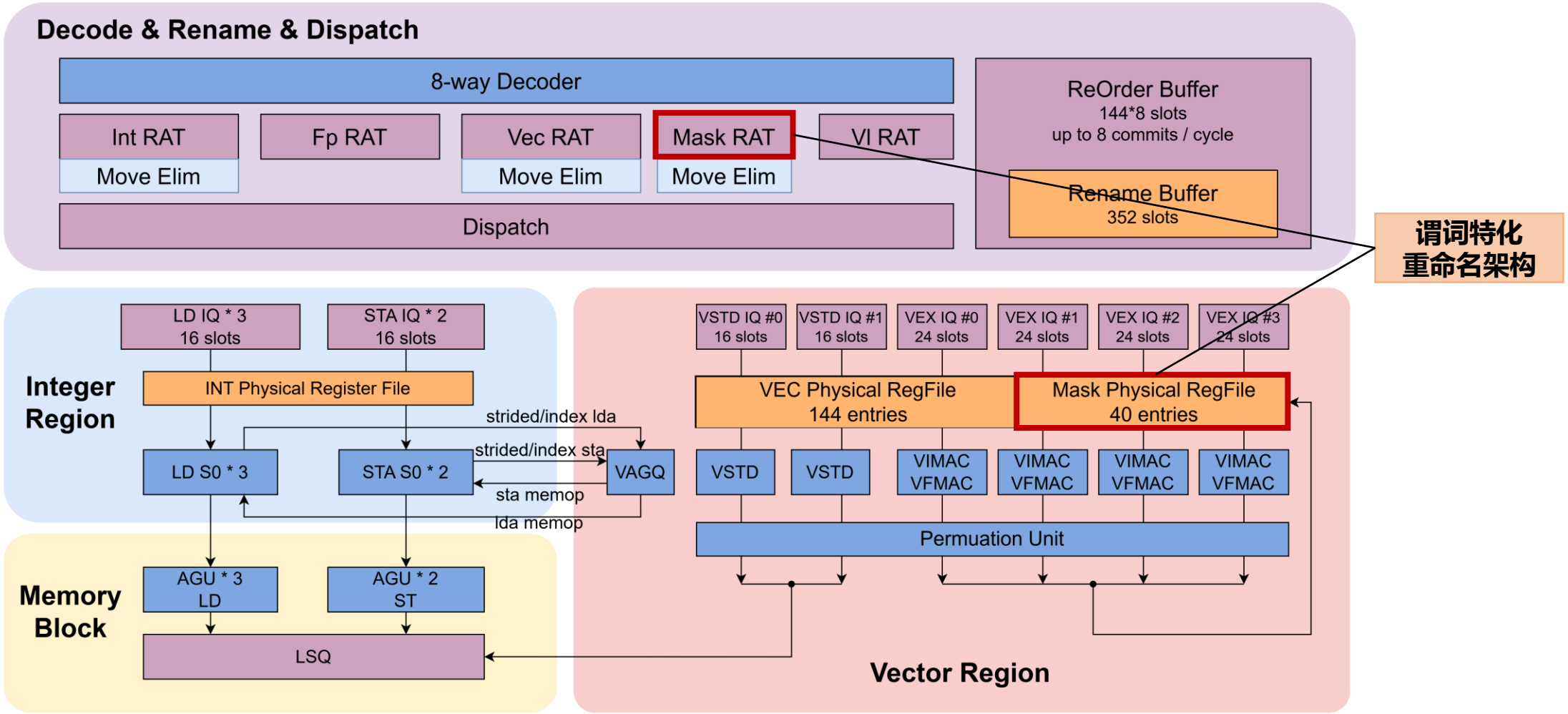
额外小优化：fault only first 指令无额外开销

向量连续load指令性能大幅提升

- vle8.v 指令吞吐提升约 60%，load to use 延迟降低至 7 周期
- vle8ff.v 指令性能与 vle8.v 相似，没有额外开销，对比 kmhv2 吞吐提升约 50 倍

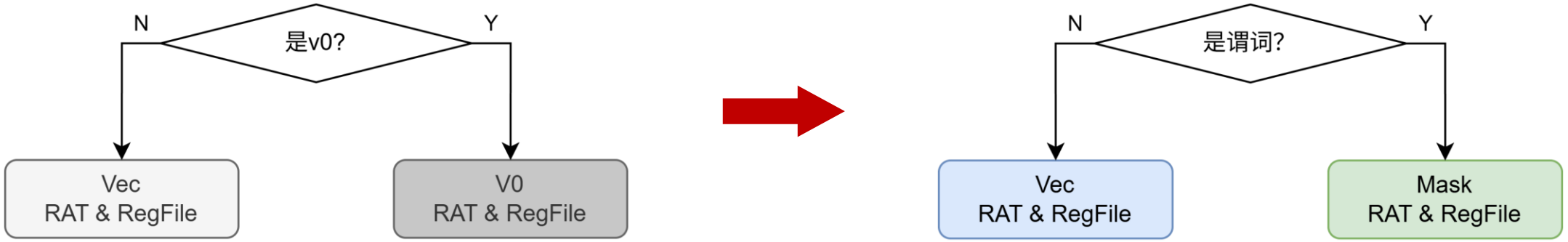


架构优化——谓词特化重命名



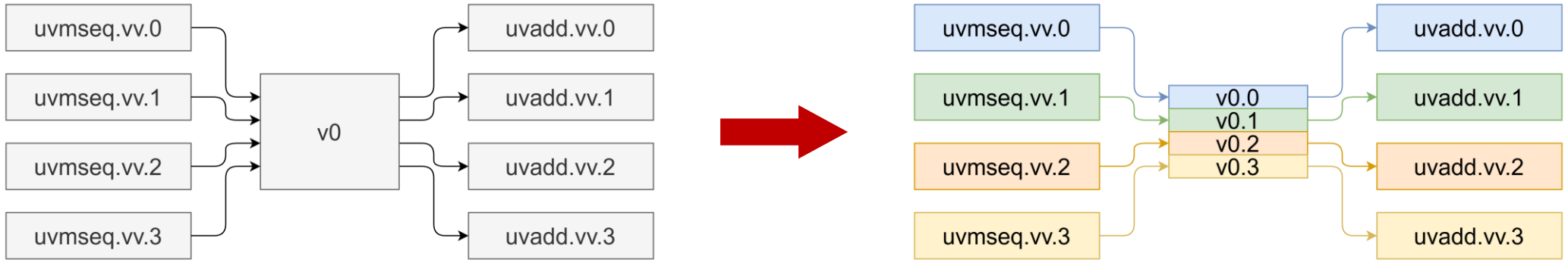
谓词寄存器堆分离优化

- kmhv2 曾使用独立 v0 RegFile 解决向量 4 操作数问题
  - vd[i] = v0.t[i] ? (vs1[i] + vs2[i]) : vd[i]
- kmhv3 基于**指令中的谓词寄存器语义**做堆分离优化，而非名称
  - vadd.vv v3, v2, v1, v0.t 等谓词化计算指令的 **v0.t 是谓词**
  - vmseq.vv v1, v0, v2 等**谓词生成指令的结果 v1 是谓词**，而 v0 是向量
  - vmand.mm v2, v1, v0 等**谓词运算的三个操作数都是谓词**



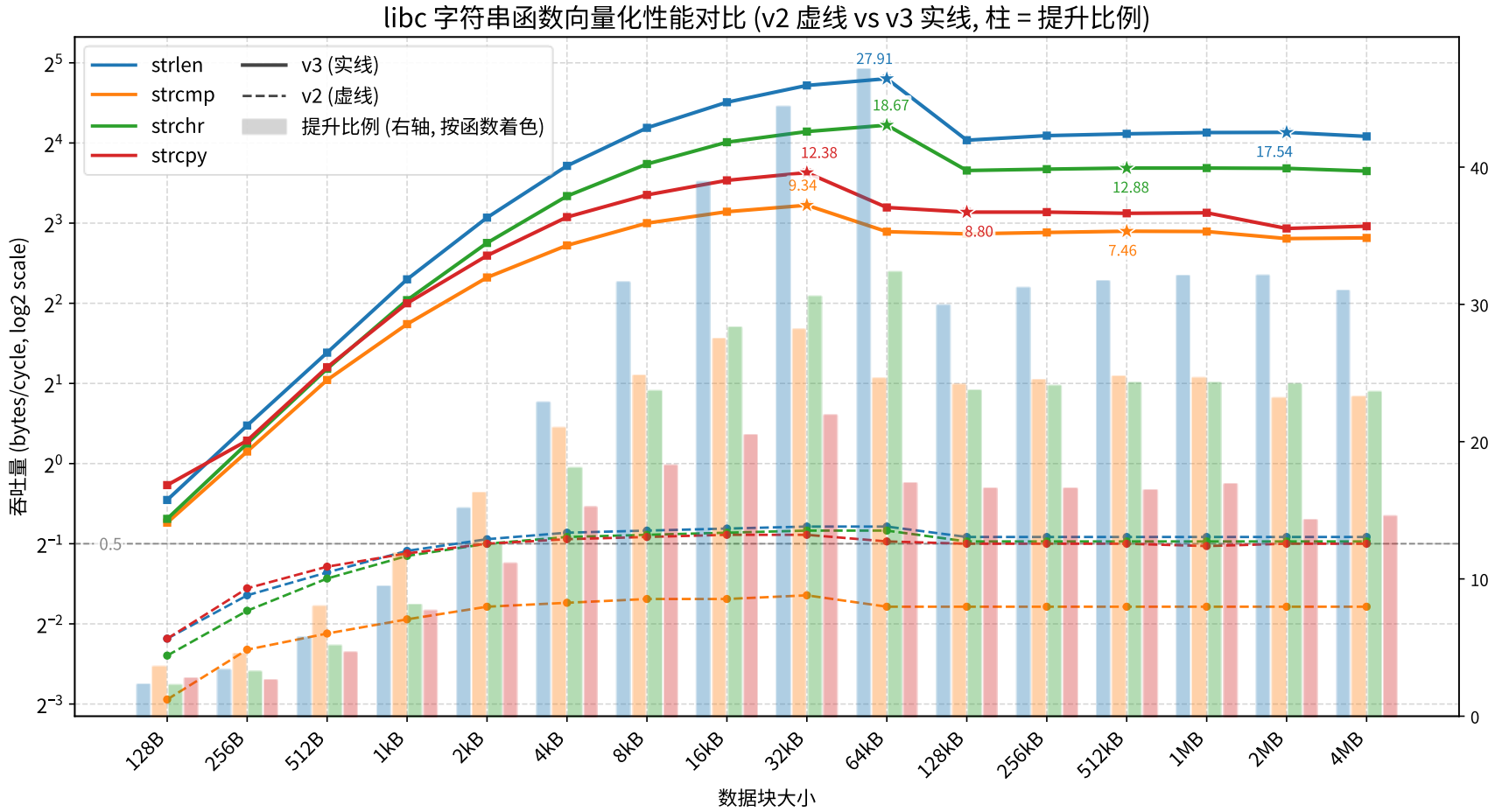
谓词优化举例

- UOP 独立的谓词寄存器编号，使得
  - LMUL>1 时谓词生产者能独立唤醒消费者，且能独立旁路数据
  - 谓词生产者能独立写回寄存器，不用拼接其它 UOP 数据
- 举例：
  - LMUL=4 时
  - 指令1 vmseq.vv v0, v4, v8 生产谓词
  - 指令2 vadd.vv v12, v16, v20, v0.t 消费谓词

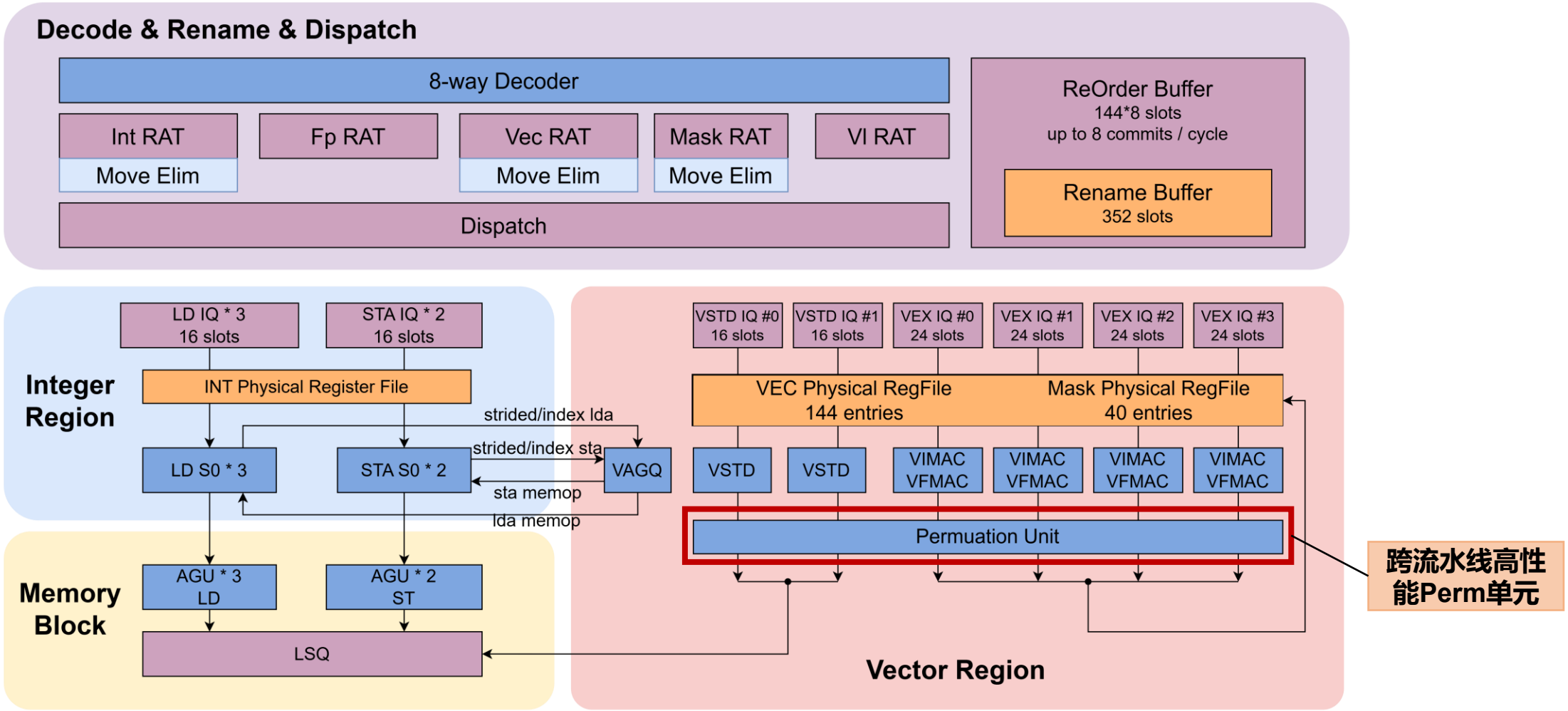


libc 字符串函数性能大幅提升

- 由译码、向量访存、谓词优化等共同贡献
- 比 kmhv2 大幅提升吞吐



架构优化——Perm单元

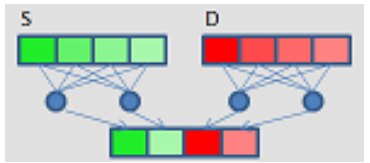

Decode & Rename & Dispatch
8-way Decoder
Int RAT
Move Elim
Fp RAT
Vec RAT
Move Elim
Mask RAT
Move Elim
VI RAT
Dispatch
ReOrder Buffer
144*8 slots
up to 8 commits / cycle
Rename Buffer
352 slots
Integer Region
LD IQ * 3
16 slots
STA IQ * 2
16 slots
INT Physical Register File
strided/index lda
strided/index sta
LD S0 * 3
STA S0 * 2
VAGQ
sta memop
lda memop
Memory Block
AGU * 3
LD
AGU * 2
ST
LSQ
VSTD IQ #0
16 slots
VSTD IQ #1
16 slots
VEX IQ #0
24 slots
VEX IQ #1
24 slots
VEX IQ #2
24 slots
VEX IQ #3
24 slots
VEC Physical RegFile
144 entries
Mask Physical RegFile
40 entries
VSTD
VSTD
VIMAC
VFMAC
VIMAC
VFMAC
VIMAC
VFMAC
VIMAC
VFMAC
Permuation Unit
Vector Region
跨流水线高性能Perm单元

RVV Perm 指令

- 包含 vrgather, vslideup, vslidedown, vcompress 等
- Segment Load/Store 辅助完成 Unzip/Zip，seg 可取 2~8
- Perm 指令广泛应用在 CPU 内数据排列算法中
- kmhv2方案：硬件替软件做循环
  - vrgather.vv 指令至多拆分 64 条 UOP
  - vcompress.vm 指令至多拆分 37 条 UOP



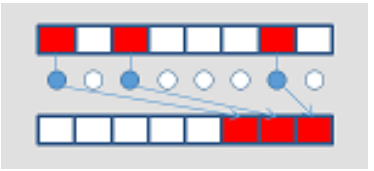
unpack



shuffle



gather



compress

各种 permutation 指令举例

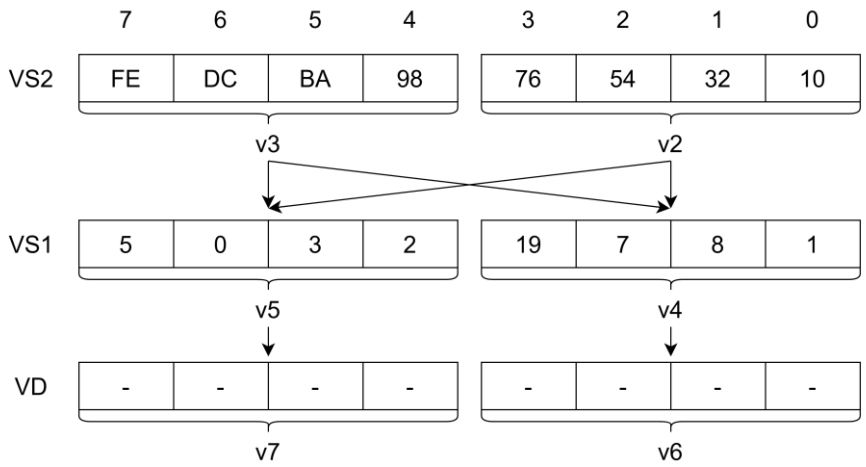
图片来源 https://www.officedaytime.com/simd512e/

vrgather 指令

- 指令实现索引取数的向量化，长向量具有数倍于短向量的算力的潜力
- 规范中定义如下

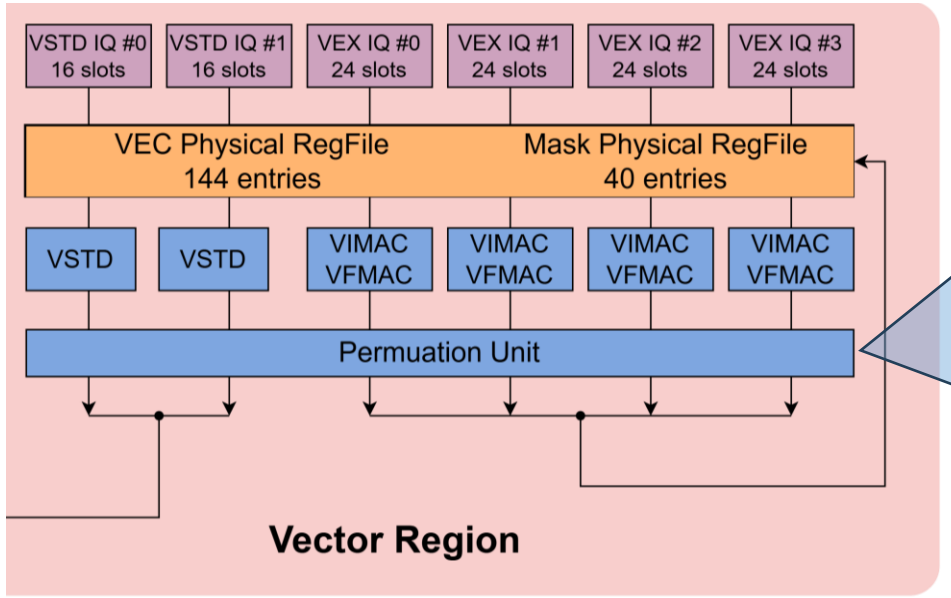
```
vrgather.vv vd, vs2, vs1, vm     # vd[i] = (vs1[i] >= VLMAX) ? 0 : vs2[vs1[i]];
vrgatherei16.vv vd, vs2, vs1, vm # vd[i] = (vs1[i] >= VLMAX) ? 0 : vs2[vs1[i]];
```

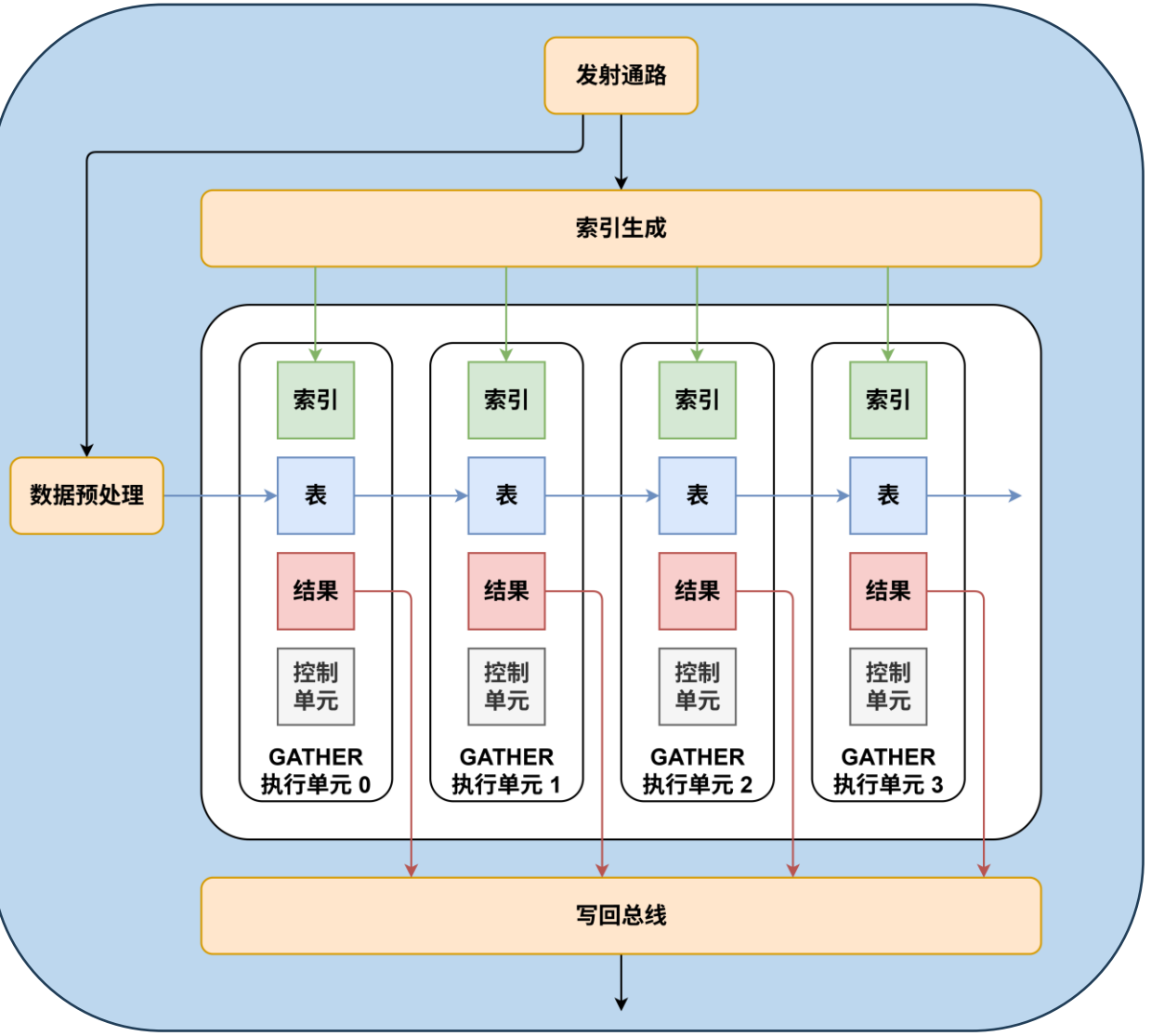
- 一个VLEN=128, LMUL=2, SEW=32的例子
  - 拆分方案需要 4 个UOP
  - 即 1 条 LMUL=2 的 vrgather 指令需要 4 条 LMUL=1 指令才能完成运算
  - LMUL=8 则对应 64 条



一维脉动阵列Perm单元

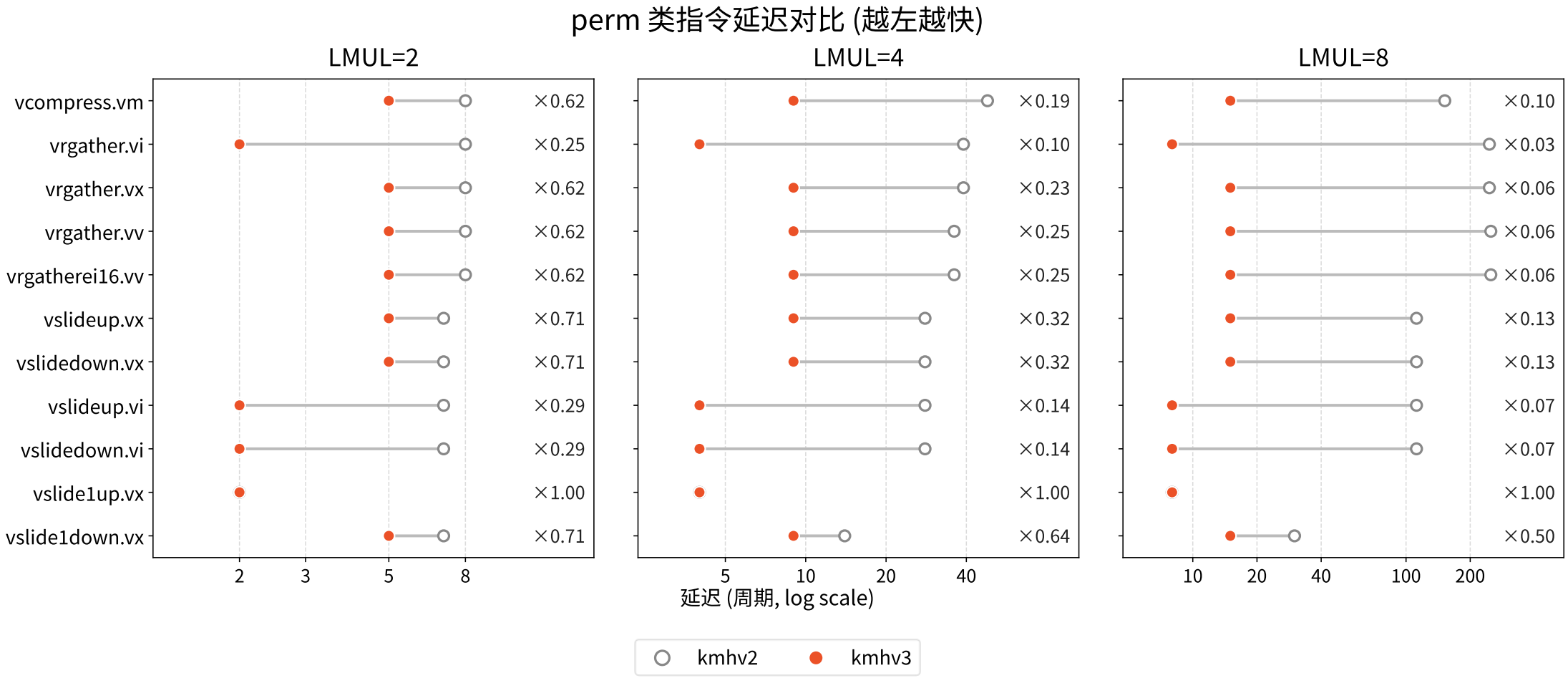
- 以 UOP 粒度接收索引元数据和表
- Perm 类型指令统一成Gather运算处理
  - 生成索引 + 数据预处理
- 发射通路和写回总线跨多条向量流水线





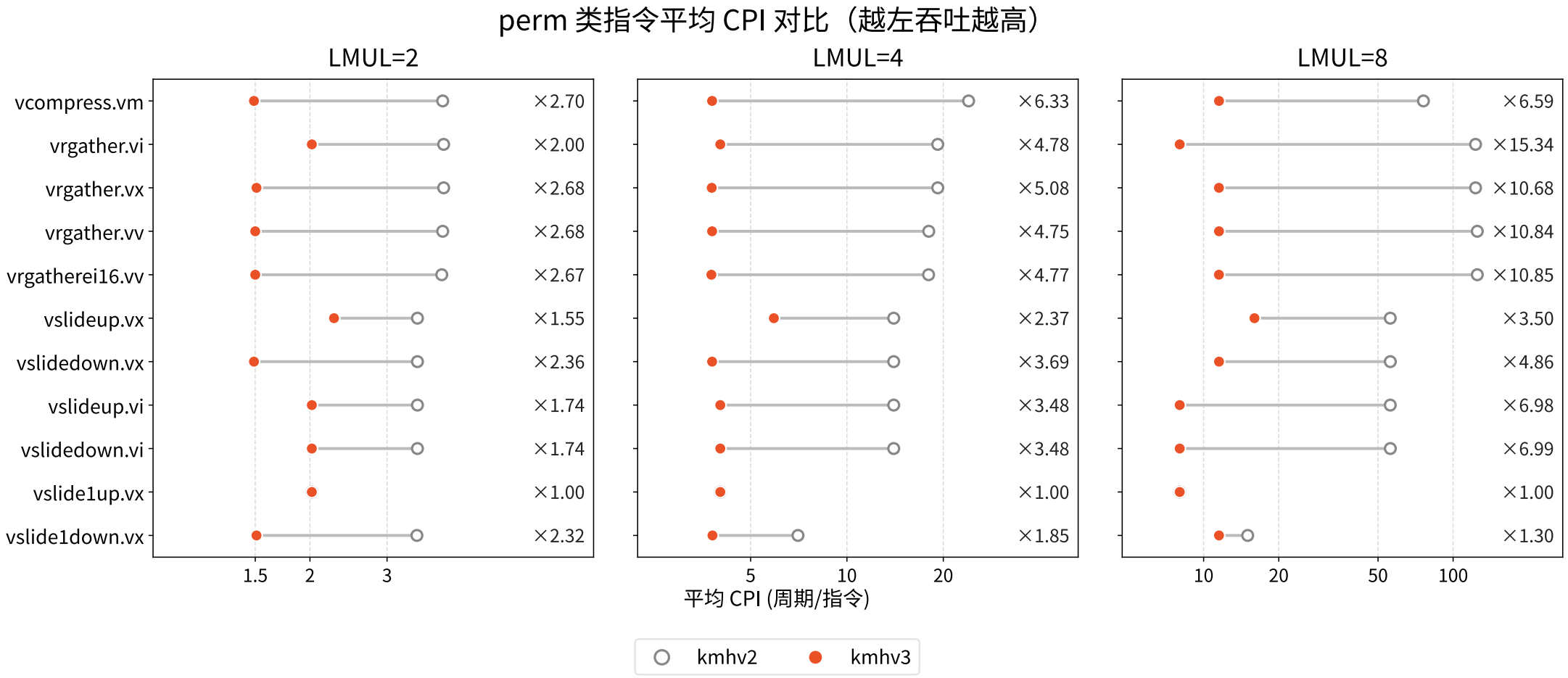
Perm 指令延迟对比

- Perm 指令从 延迟 $\propto$ LMUL^2 降低至 延迟 $\propto$ LMUL



Perm 指令吞吐对比

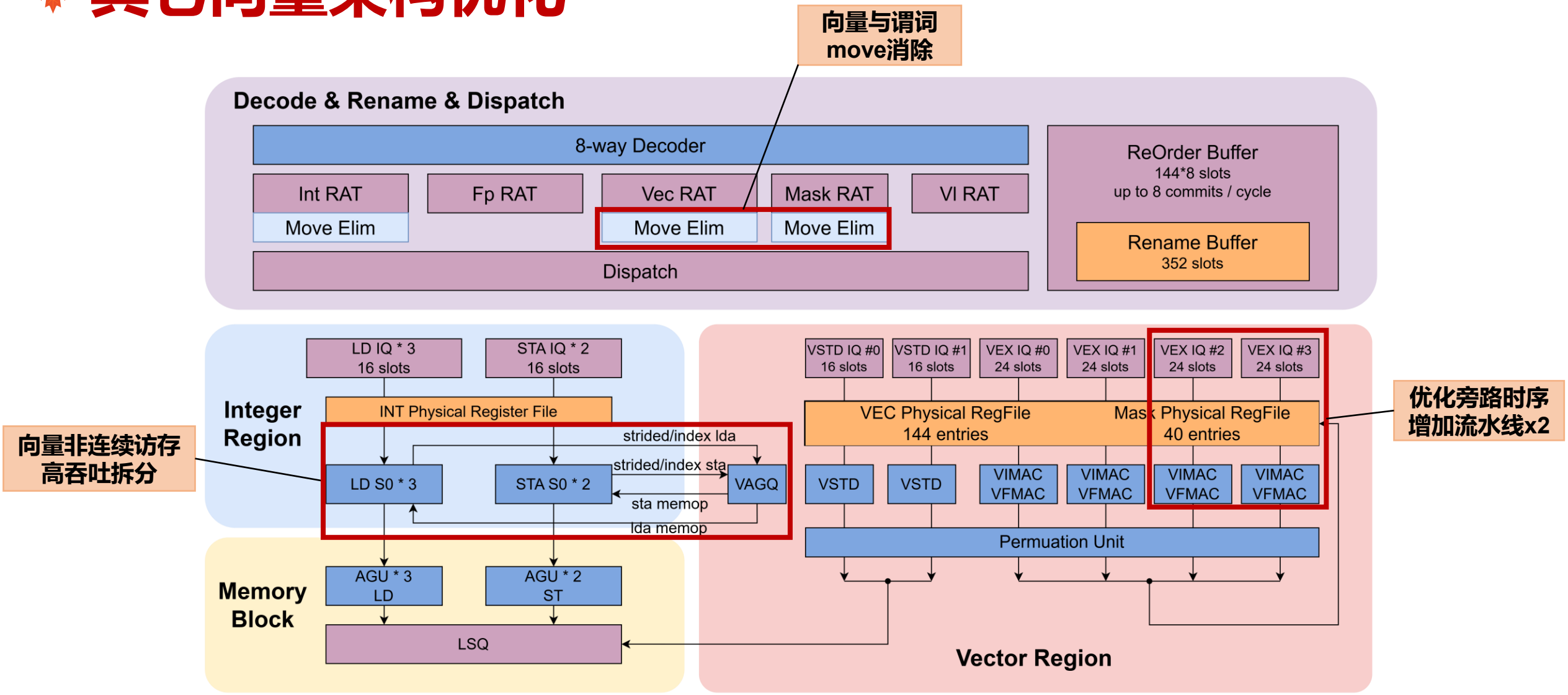
- Perm 指令吞吐也有数倍提升



Perm 小结

- 香山的 Perm 单元延迟和吞吐都有极大优化
- LMUL=8 的 Perm 运算收益高于 3组 8寄存器溢出到内存的代价
- **能在 CPU 内重排就不要访存**

其它向量架构优化


向量与谓词
move消除
Decode & Rename & Dispatch
8-way Decoder
Int RAT
Move Elim
Fp RAT
Vec RAT
Move Elim
Mask RAT
Move Elim
VI RAT
Dispatch
ReOrder Buffer
144*8 slots
up to 8 commits / cycle
Rename Buffer
352 slots
向量非连续访存
高吞吐拆分
Integer
Region
LD IQ * 3
16 slots
STA IQ * 2
16 slots
INT Physical Register File
strided/index lda
strided/index sta
LD S0 * 3
STA S0 * 2
VAGQ
sta memop
lda memop
Memory
Block
AGU * 3
LD
AGU * 2
ST
LSQ
VSTD IQ #0
16 slots
VSTD IQ #1
16 slots
VEX IQ #0
24 slots
VEX IQ #1
24 slots
VEX IQ #2
24 slots
VEX IQ #3
24 slots
VEC Physical RegFile
144 entries
Mask Physical RegFile
40 entries
VSTD
VSTD
VIMAC
VFMAC
VIMAC
VFMAC
VIMAC
VFMAC
VIMAC
VFMAC
Permuation Unit
Vector Region
优化旁路时序
增加流水线x2

其它向量架构优化

- **向量&谓词寄存器 move 消除**
  - 每周期支持 8 uop 消除
- **向量非连续访存指令**
  - 每周期可生成 3 个 load 或 2 个 store 请求
  - vlse8.v CPI 从 LMUL*16 降低至 LMUL*6
- **向量 Segment 访存指令**
  - vlseg 指令 CPI 降低至 2 * seg，每两周期执行一条 vlseg2e8.v 指令
  - 吞吐大幅提升
- **增加 2 条运算流水线**
  - 大幅优化数据旁路时序
  - 支持 vadd 指令 1 周期背靠背唤醒

部分SPEC06子项性能预览

- 基线SHA：09453ff8f
- 编译器版本：gcc16.1(g6afcc4f6d)
- 编译参数：
  -march=rva23u64 -ffast-math -flto -ffp-contract=off -ftree-vectorize \
  -mrvv-max-lmul=**m8** -mrvv-vector-bits=zvl -fno-strict-aliasing
- 内存库：jemalloc
- 评估方式：gsim 软件切片仿真

子项	基线分数	向量分数	性能增减因素
429.mcf	26.015	26.957(+3.62%)	向量 libc mem 函数
456.hmmer	18.271	23.978(+31.24%)	整数SIMD与reduction
464.h264ref	23.133	26.234(+13.41%)	segment 访存和 perm 运算
483.xalancbmk	29.431	33.463(+13.70%)	向量 libc str 函数
454.calculix	13.968	6.644(-52.43%)	大LMUL导致向量寄存器频繁溢出到内存 大量标量拼向量指令，未考虑slide代价

一些愿望

- **对香山**
  - 新向量代码在未来 3-4 个月陆续开源合入主线（9月起已有代码合入）
  - xs-gem5 对齐新向量架构设计
  - 评估其它 LMUL 配置及 llvm 与 xscc 编译器的向量化收益
- **对 RVV 生态**
  - 编译器考虑寄存器溢出的动态 LMUL 设定
  - 拥抱香山，软硬件协同探索极致的通用计算性能

总结

- 无空泡向量标量混合译码
- 向量连续访存降低延迟、提升吞吐
- 高性能 Perm 运算单元
- 语义级的谓词调度
- move 消除、时序优化、流水线增加等优化
- **香山的向量正在变得好用**



感谢聆听