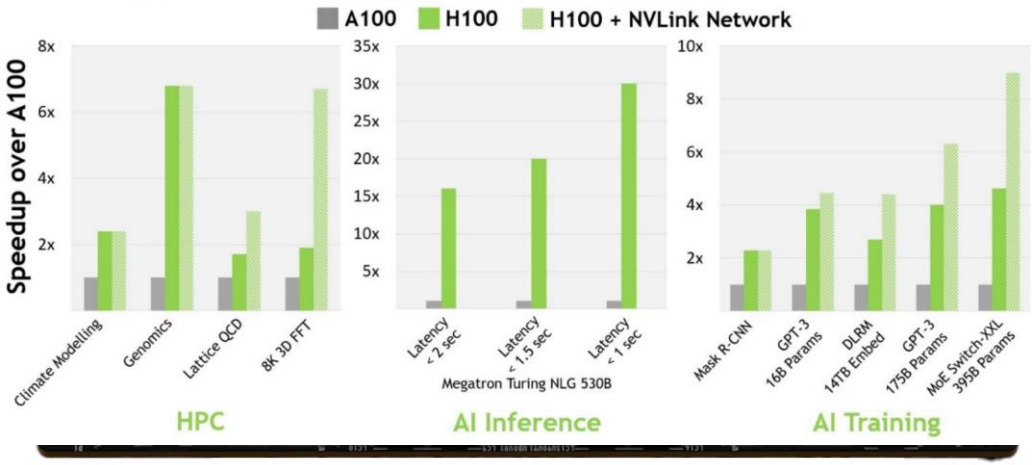
RISC-V向量扩展与GPGPU的结合：以“承影”为例

杨轲翔

清华大学集成电路学院

背景 - GPGPU

- GPU：采用并行计算的方式同时进行大量运算
- GPGPU：具备通用计算能力的高吞吐率处理器
  - AI领域
  - 高性能计算领域
- 开源GPU (GPGPU)项目
  - 英属哥伦比亚 GPGPU-Sim
  - 威斯康星麦迪逊 MIAOW GPGPU
  - 乔治亚理工 Vortex GPU

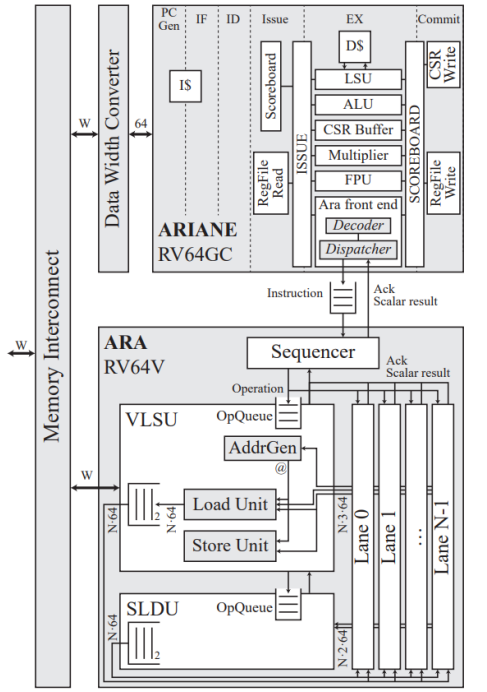


*图片来自Nvidia H100 Tensor Core GPU Architecture

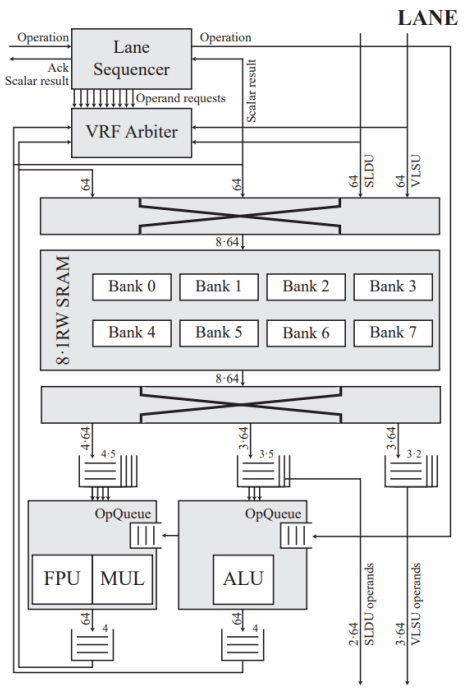
背景 - RISC-V向量扩展

- RISC-V向量扩展(RVV)的发展
- 2021年9月正式推出1.0版本

- pulp-platform开源RVV处理器Ara
- GCC和LLVM已经支持RVV汇编
- 晶心科技NX27V向量处理器
- 兆松科技基于LLVM框架，实现了一套高质量的RVV自动向量化工具



(a) Block diagram of an Ara instance with *N* parallel lanes. Ara receives its commands from Ariane, a RV64GC scalar core. The vector unit has a main sequencer; *N* parallel lanes; a Slide Unit (SLDU); and a Vector Load/Store Unit (VLSU). The memory interface is *W* bit wide.

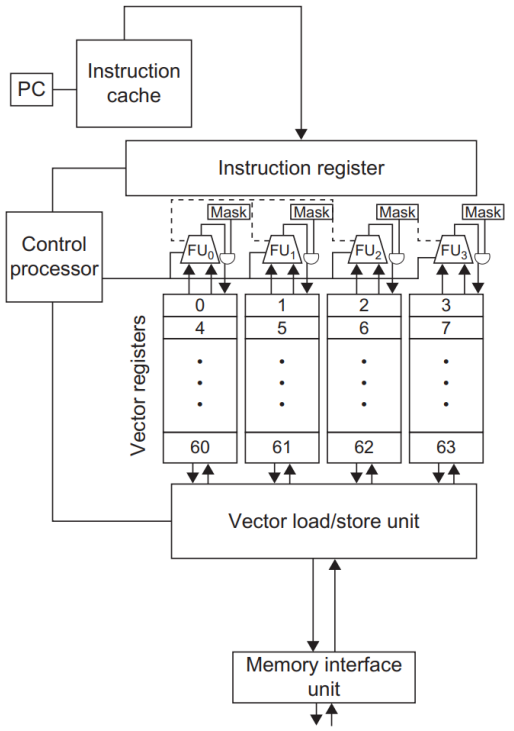


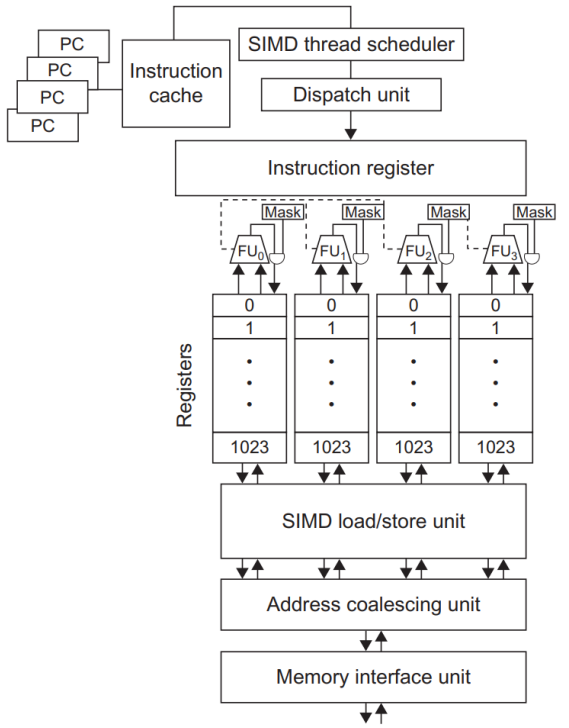
(b) Block diagram of one lane of Ara. It contains a lane sequencer (handling up to 8 vector instructions); a 16 KiB vector register file; ten operand queues; an integer Arithmetic Logic Unit (ALU); an integer multiplier (MUL); and a Floating Point Unit (FPU).

*图片来自Ara: A 1-GHz+ Scalable and Energy-Efficient RISC-V Vector Processor With Multiprecision Floating-Point Support in 22-nm FD-SOI

向量处理器与GPGPU

- 典型向量处理器与典型SIMT架构在计算、访存、分支、任务控制、寄存器堆较为相似
- 动机：将向量lane与GPGPU thread一一对应



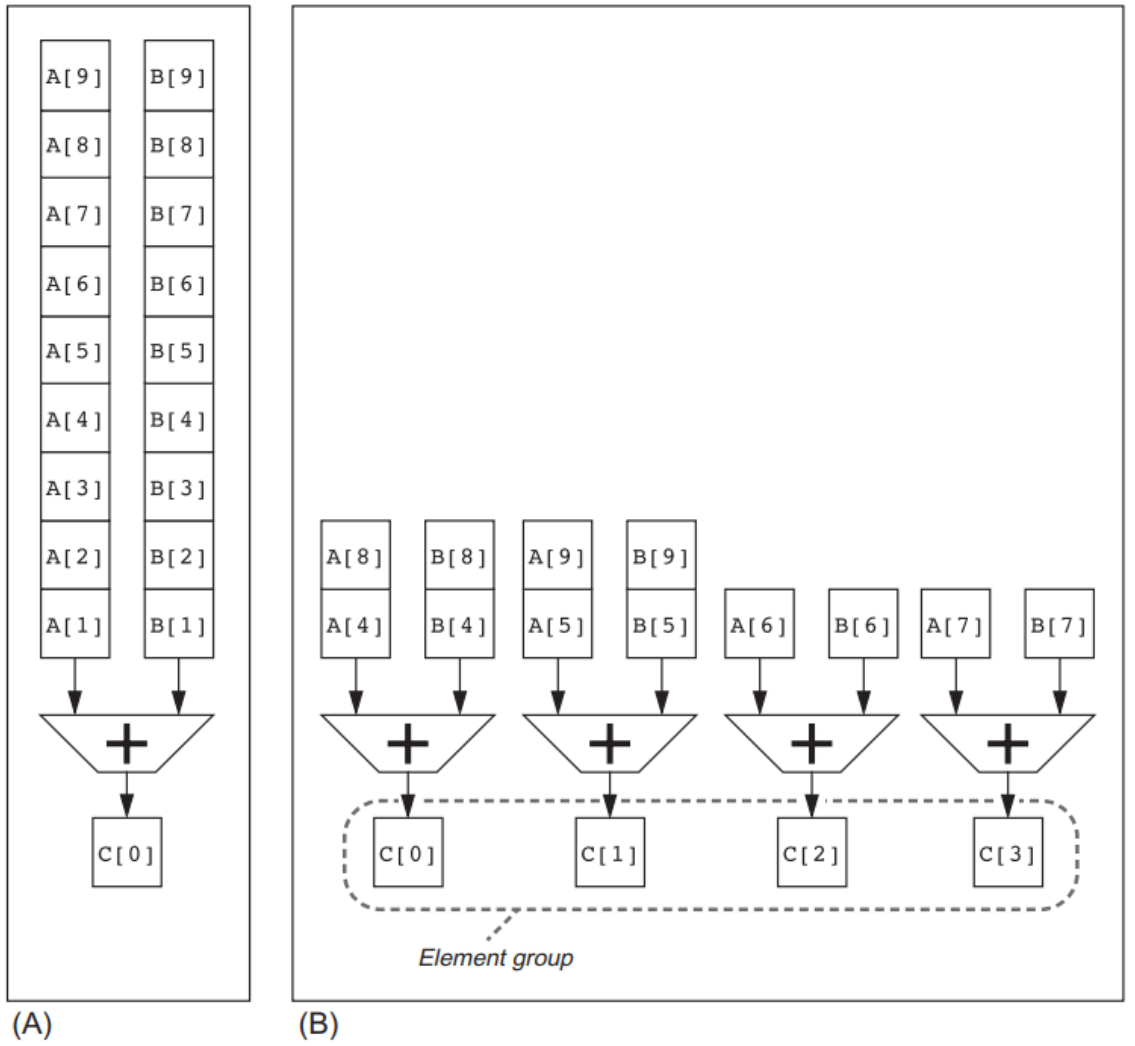


	计算单元	访存单元	分支控制	任务控制	寄存器堆	整体结构
Vector	Vector Lane	Gather/Scatter	Mask Registers	Control Processor	Vector Registers	Vector Processor
GPGPU	SIMD Lane	Global load/store	Predicate Registers	Thread Block Scheduler	SIMD Lane Registers	Multithreaded SIMD Processor

*图片来自《计算机体系结构 量化研究方法》

向量处理器与GPGPU

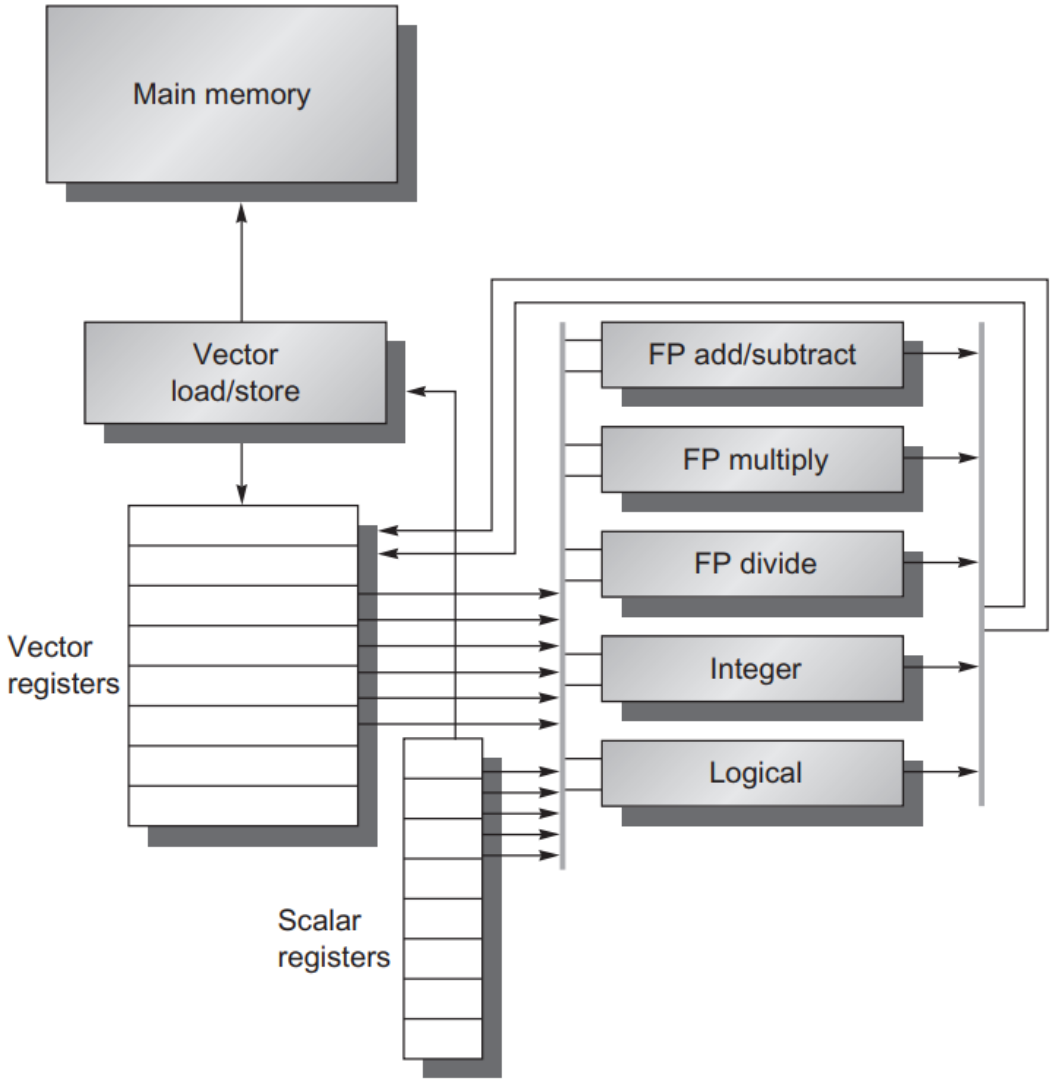
- RISC-V向量扩展优点：
  - 适应不同规模的硬件，无需重新编译
  - 具备scalarization
  - 访存特性表征
- SIMT架构优点：
  - warp级别并行调度
  - 编程灵活性高，对程序员友好



*图片来自《计算机体系结构 量化研究方法》

向量处理器与GPGPU

- RISC-V向量扩展优点：
  - 适应不同规模的硬件
  - 具备scalarization：用scalar寄存器和scalarALU计算公共数据地址
  - 访存特性表征
- SIMT架构优点：
  - warp级别并行调度
  - 编程灵活性高，对程序员友好

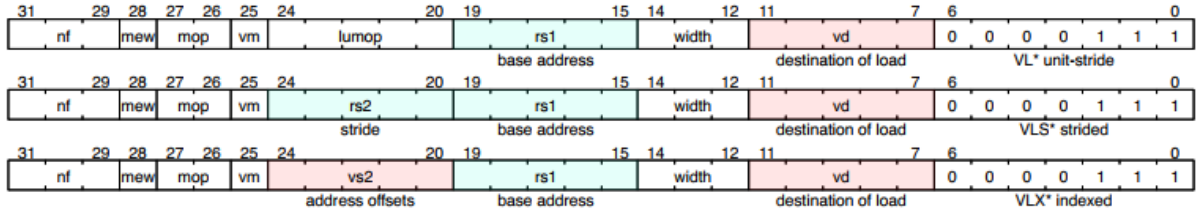


*图片来自《计算机体系结构 量化研究方法》

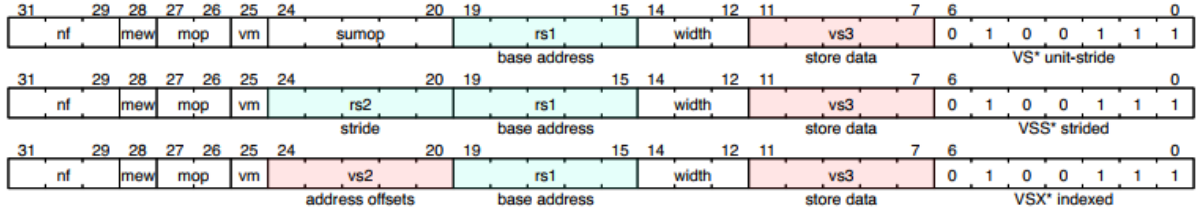
向量处理器与GPGPU

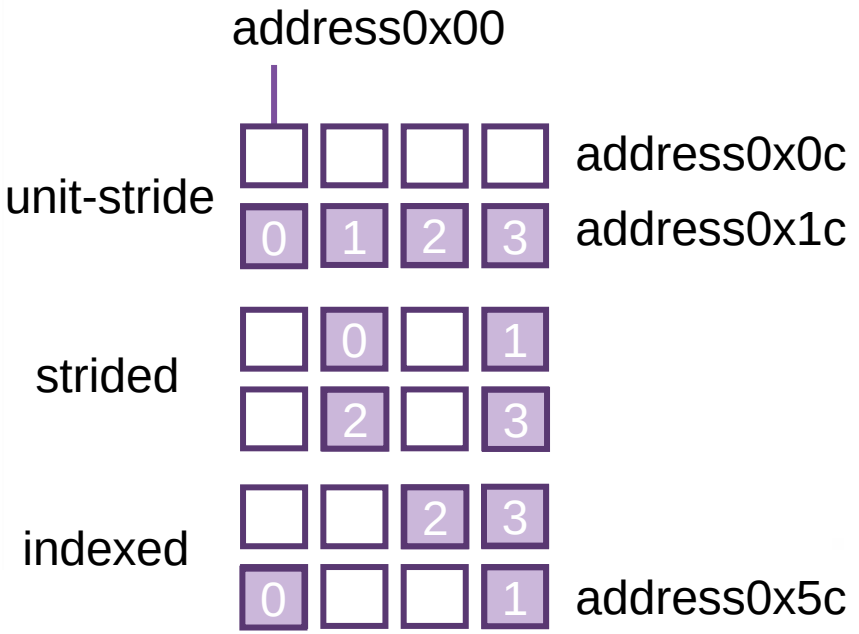
- RISC-V向量扩展优点：
  - 适应不同规模的硬件
  - 具备scalarization
  - 访存特性表征：指令中指示了访存类型：连续、步进和索引
- SIMT架构优点：
  - warp级别并行调度
  - 编程灵活性高，对程序员友好

Format for Vector Load Instructions under LOAD-FP major opcode



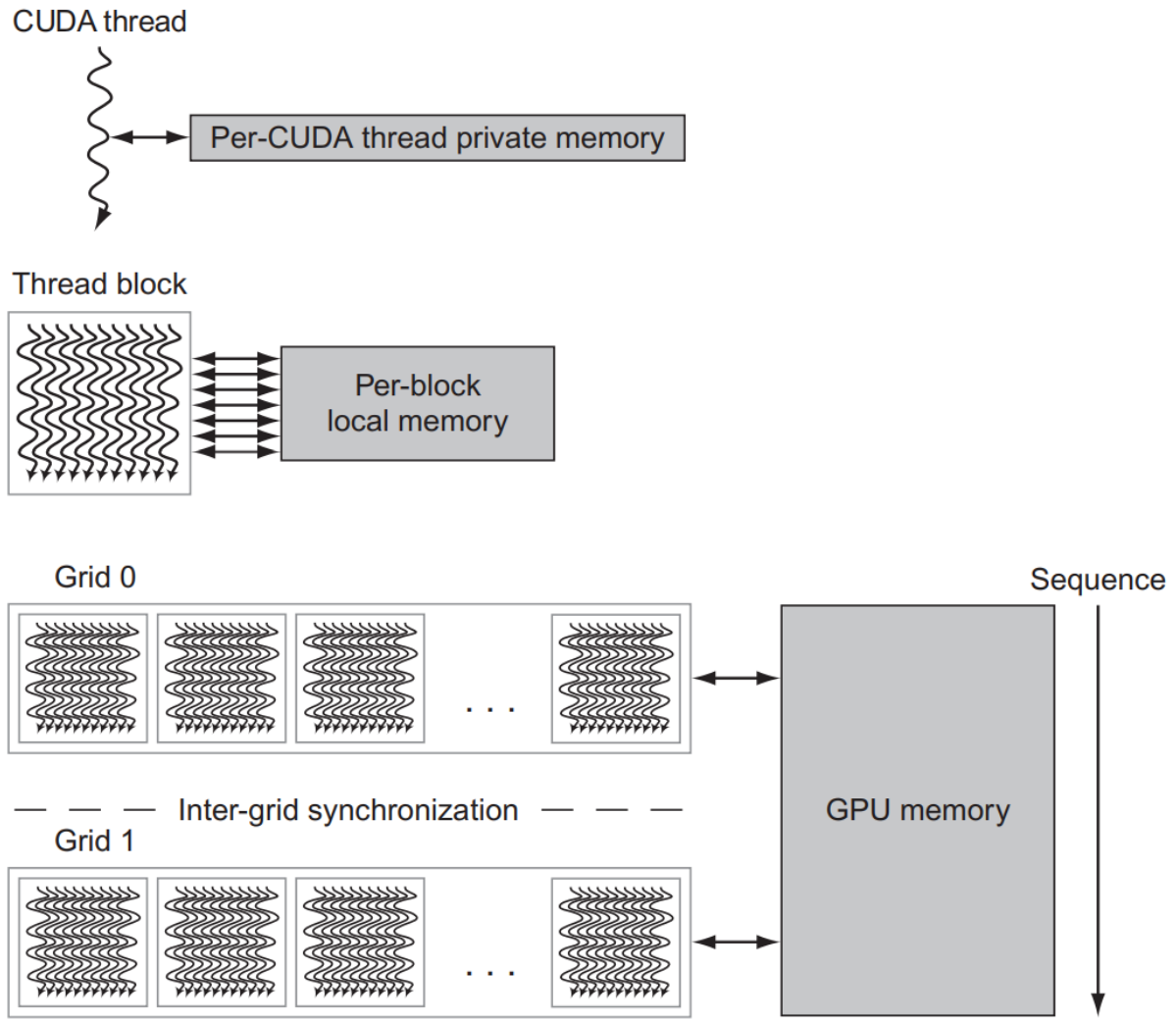
Format for Vector Store Instructions under STORE-FP major opcode





向量处理器与GPGPU

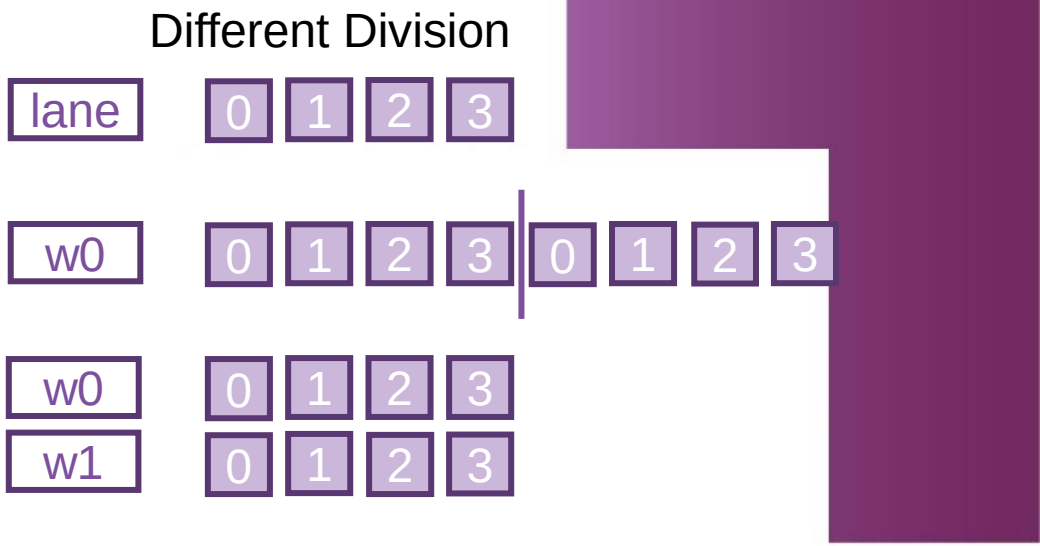
- RISC-V向量扩展优点：
  - 适应不同规模的硬件
  - 具备scalarization
  - 访存特性表征
- SIMT架构优点：
  - warp级别并行：硬件将thread按32个分为一组整体执行，通过切换warp掩盖延时
  - 编程灵活性高，对程序员友好：CUDA编程仅需针对单个thread行为进行描述

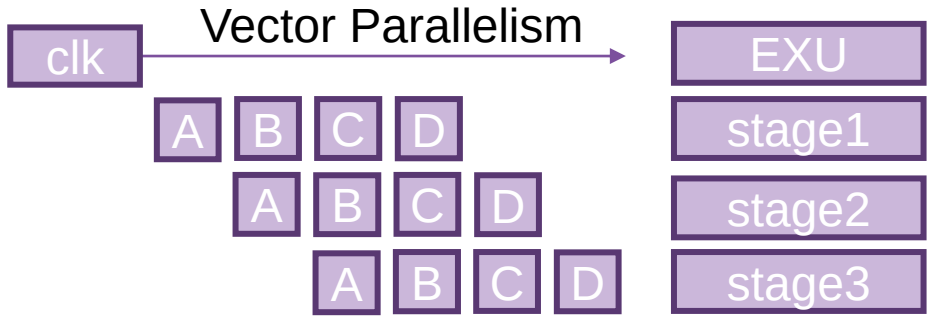


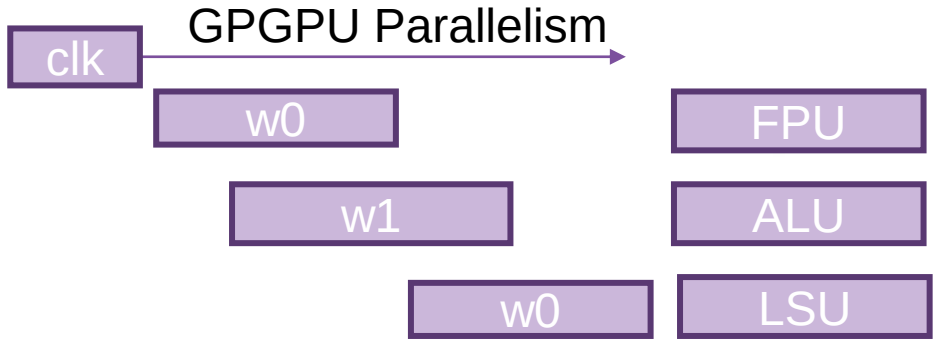
*以下用warp表示一段SIMT/RVV程序，是硬件调度和执行程序段的最小单位
*图片来自《计算机体系结构 量化研究方法》

向量处理器与GPGPU

- 灵活裁剪：单程序内的向量长度可在warp和向量循环间灵活划分，从而探索不同层次的并行性：
  - 向量处理器的并行：单条向量指令的向量元素间
  - GPGPU的并行：同时运行的warp间
- 可根据应用程序pattern、可用硬件资源等采用不同的裁剪方式。





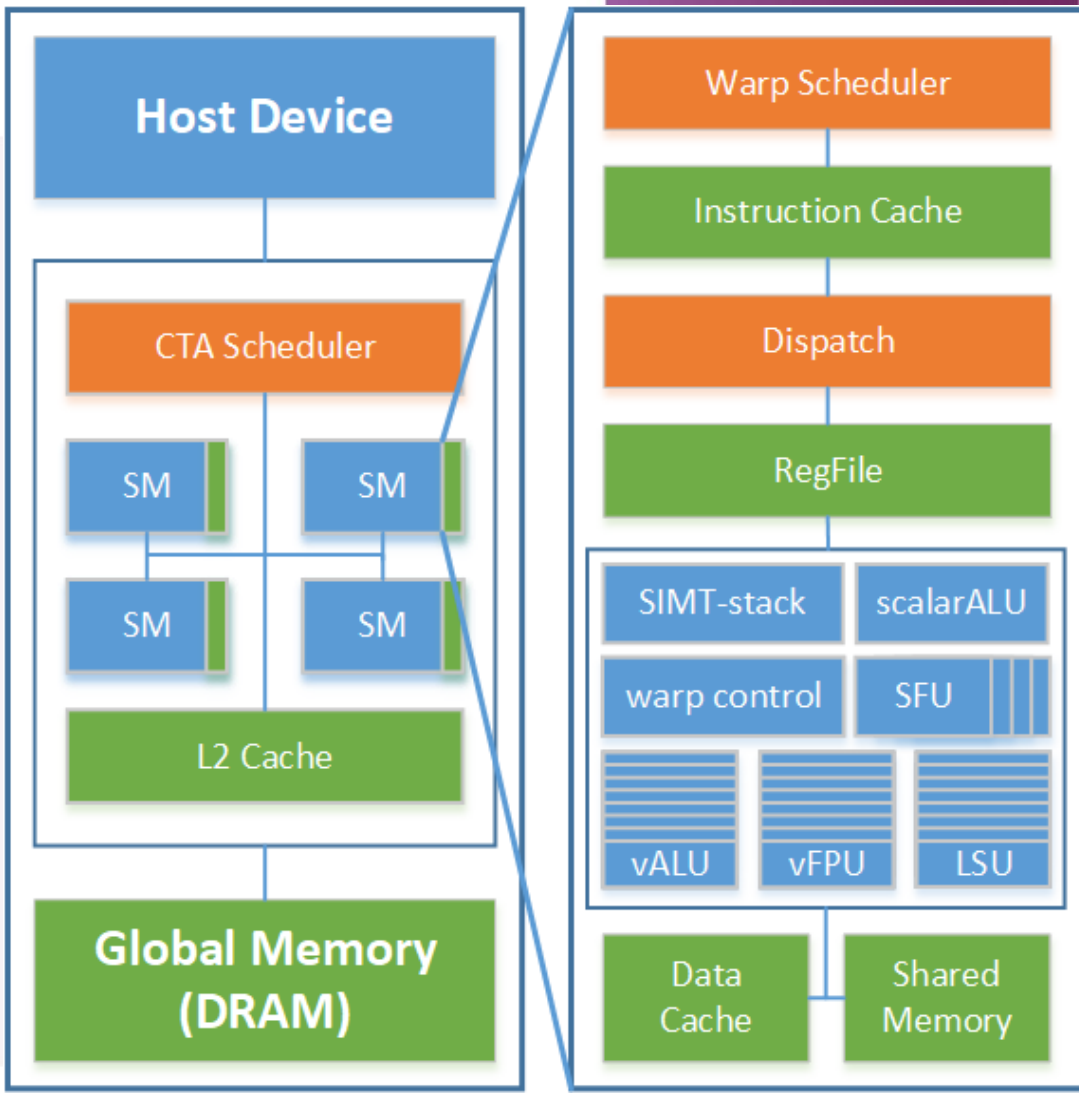


向量处理器与GPGPU

- “承影” GPGPU将向量与SIMT结合：
  - 在向量处理器的基础上提供warp粒度的并行扩展性，供程序员/编译器灵活选择，裁剪向量
  - 在GPGPU的基础上具备Scalarization、可适配不同数目功能单元、指令标记访存特性
- 采用RISC-V向量扩展，兼容开源工具链完成GPGPU编译器的开发
- RVV GPGPU与RISC-V CPU结合，完成统一指令集架构SoC系统，能让编译器在任务划分和协同上探索更多可能，寻找更佳的优化空间
- 借助开源RISC-V、开源硬件，打造自主可控的SoC系统

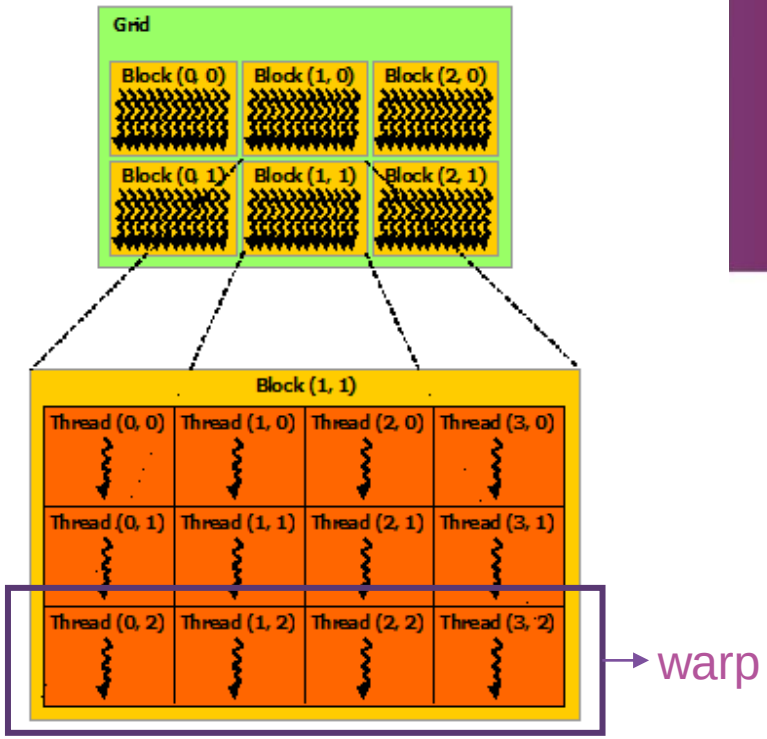
“承影” GPGPU微架构

- “承影”GPGPU硬件实现方案：
- GPGPU编程模型
- 任务分配
- 任务执行



“承影” GPGPU微架构

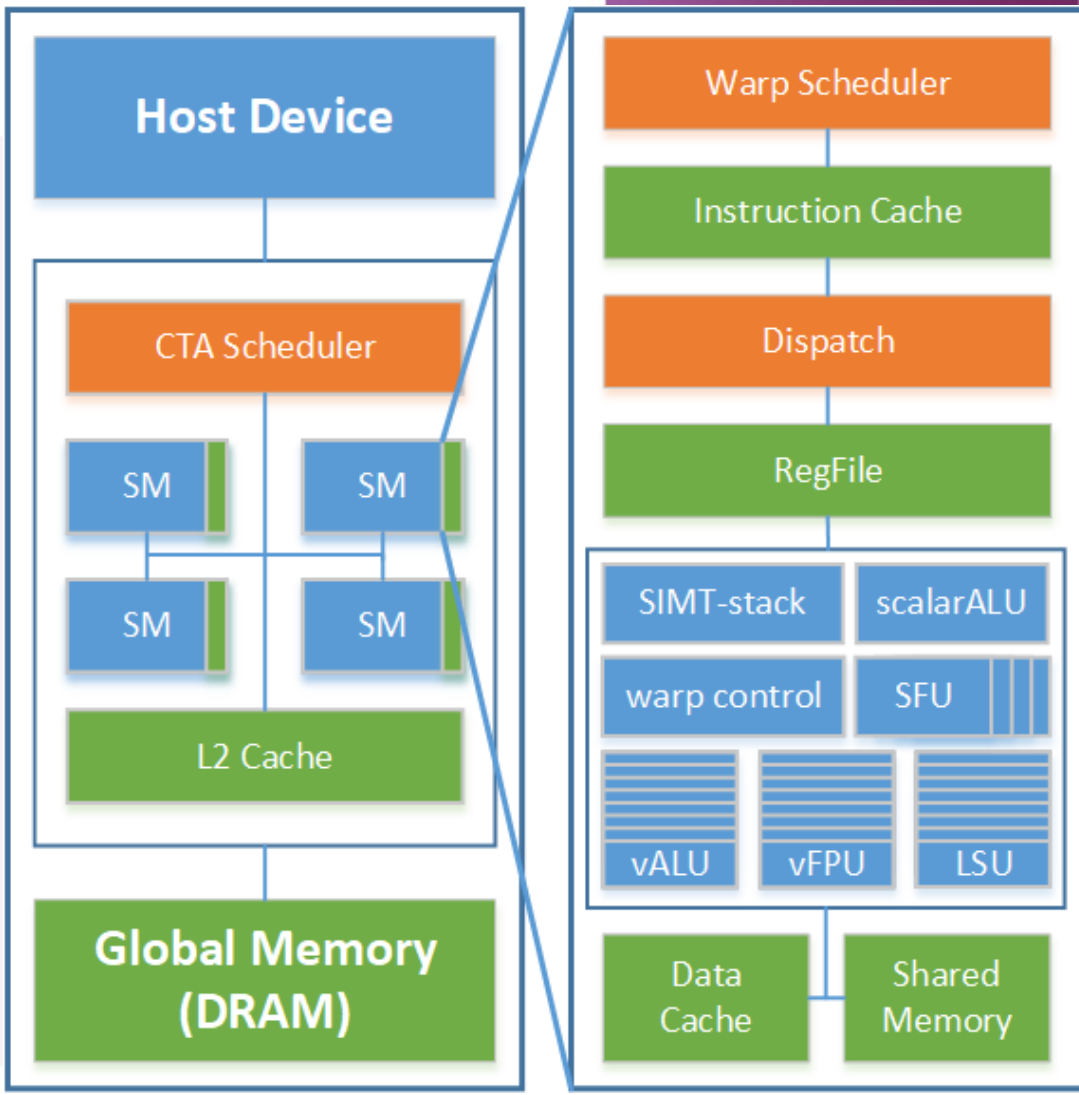
- GPGPU编程模型：
  - grid – block(CTA) – thread
  - 程序员声明需要的thread数目，然后对单个thread的行为进行描述，实现并行编程
  - warp是一定数目的thread，硬件将block中的thread组织起来，以warp为单位映射到硬件上执行
- 任务分配
- 任务执行



*图片来自Programming Guide :: CUDA Toolkit Documentation (nvidia.com)

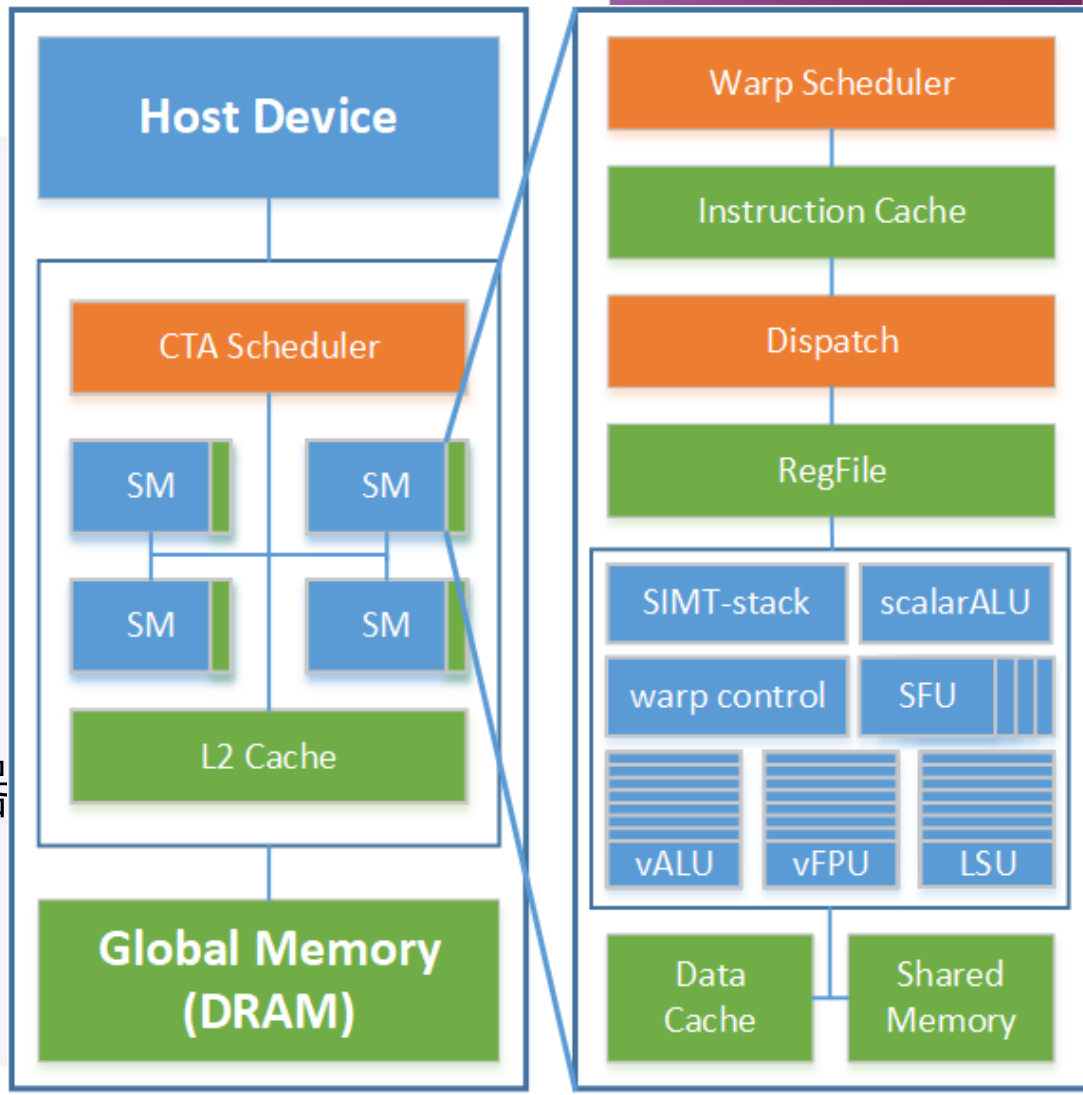
“承影” GPGPU微架构

- GPGPU编程模型
- 任务分配：Host - CTA Scheduler – SM(streaming multiprocessor)
  - host将任务以block为单位发给GPGPU
  - CTA Scheduler以warp为单位发给SM
- 任务执行



“承影” GPGPU微架构

- GPGPU编程模型
- 任务分配：Host - CTA Scheduler - SM(streaming multiprocessor)
- 任务执行：
  - 每个SM可视为一个支持多warp调度的RISC-V向量处理器
  - 每个warp可视为一段RVV程序，经由取指、译码、发射，执行后写回寄存器



“承影” GPGPU微架构 - 指令集

- ISA：RV32IMF+V
- 支持RVV指令一览
  - VLEN=num_thread*32b, ELEN=32b, LMUL=1
  - 涉及数据位宽变动的指令暂不支持，仅保留vset{i}vl{i}
  - 考虑SIMT编程的特殊性，目前优先实现独立数据通路的指令

扩展	指令类型	1.0版本	2.0版本
V	Configuration-Setting	部分支持	部分支持
	Loads and Stores	支持	支持
	Integer Arithmetic	支持	支持
	Floating-Point	支持	支持
	Reduction	不支持	参照GPGPU需求添加
	Mask	部分支持	参照GPGPU需求添加
I		部分支持	添加对异常处理的支持
M		支持	支持
F		支持	支持
A		不支持	支持

“承影” GPGPU微架构 - 指令集

- 自定义指令：
  - 分支与汇合
  - 同步
  - warp结束
- 后续还将添加…
  - dynamic parallelism

	31 25	24 20	19 15	14 12	11 7	6 0	assemble	description
Custom	off[12|10:5]	vs2	vs1	0 0 0	off[4:1|11]	0 0 0 1 0 1 1	vbeq vs2, vs1, offset	if(vs2 == vs1) PC(in_stack) += sext(offset)
				0 0 1			vbne vs2, vs1, offset	if(vs2 != vs1) PC += sext(offset)
				1 0 0			vblt vs2, vs1, offset	if(vs2 <s vs1) PC += sext(offset) //signed
				1 0 1			vbge vs2, vs1, offset	if(vs2 >=s vs1) PC += sext(offset) //signed
				1 1 0			vbltu vs2, vs1, offset	if(vs2 <u vs1) PC += sext(offset) //unsigned
				1 1 1			vbgeu vs2, vs1, offset	if(vs2 >=u vs1) PC += sext(offset) //unsigned
	off[12|10:5]	vs2	vs1	0 1 1	off[4:1|11]	0 0 0 1 0 1 1	join vs2,imm,offset	join
	0 0 0 0 0 0 0	rs2	rs1	0 1 0	rd	0 0 0 1 0 1 1	endprg	reach the end of the program
	0 0 0 0 0 0 1	vs2	imm[4:0]	0 1 0	rd	0 0 0 1 0 1 1	barrier x0	reach the barrier

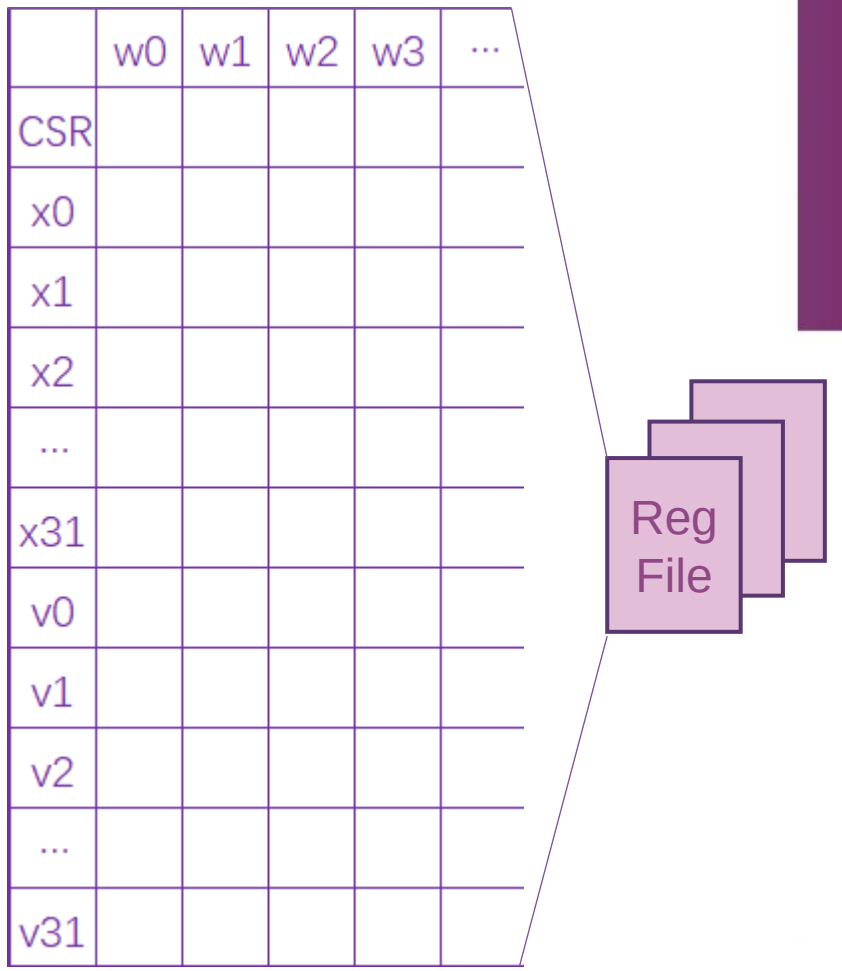
“承影” GPGPU微架构 - 寄存器堆

每个warp有

- 32 * 32bit scalarReg
- 32 * num_thread * 32bit vectorReg
- CSR：保留当前warp信息

当前版本设计中由RISC-V限定寄存器数目，不动态分配

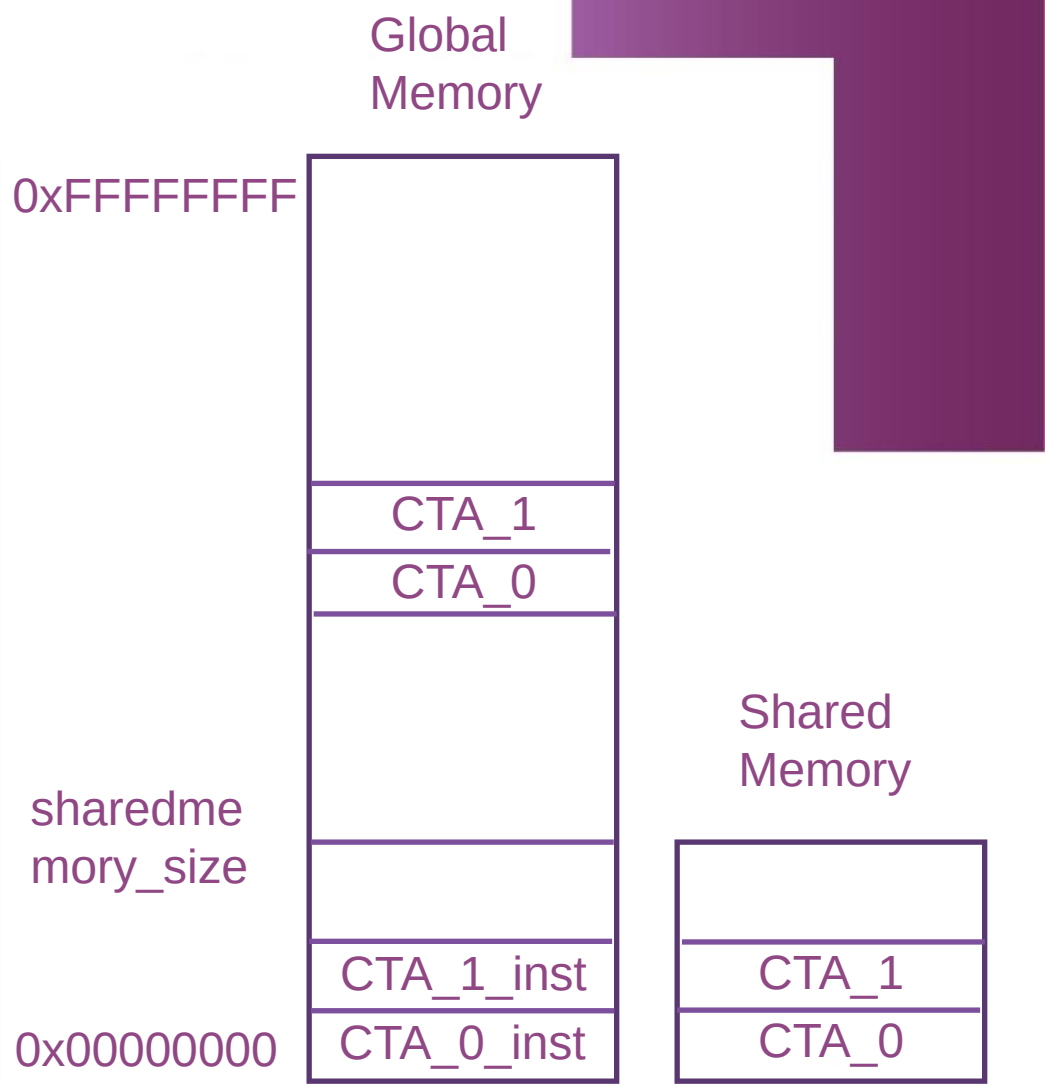
若未来有软件支持，可在相同寄存器堆大小下分配更多的warp



“承影” GPGPU微架构 - 存储模型

Memory Model

- 软件实现地址空间划分：
  - warp CSR配置基址信息
  - SharedMemory（数据）
  - GlobalMemory（数据、指令）
- warp数据同步：
  - 软件插入同步标记
  - 硬件barrier执行同步功能
  - 未来考虑使用RVWMO完善同步模型

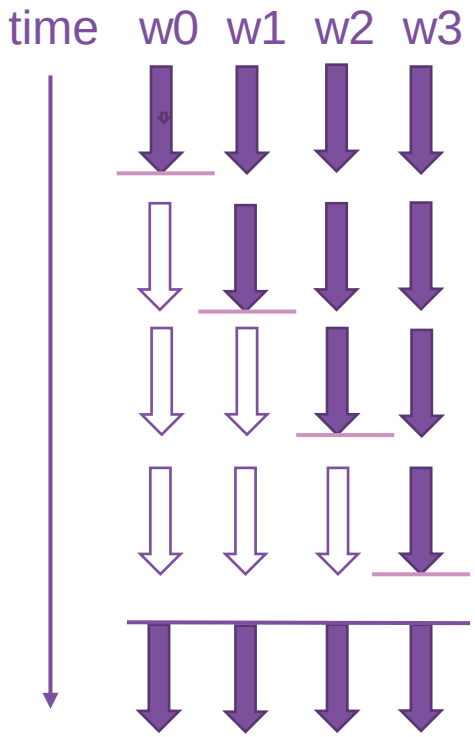


“承影” GPGPU微架构 - 存储模型

Memory Model

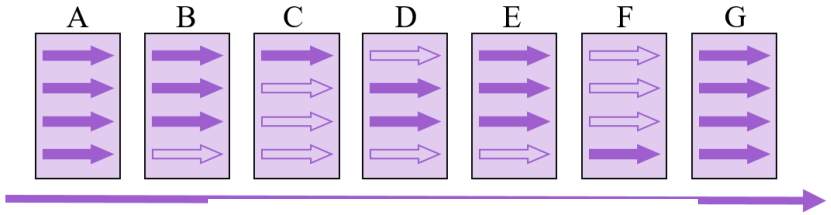
- 软件实现地址空间划分：
  - warp CSR配置基址信息
  - SharedMemory（数据）
  - GlobalMemory（数据、指令）
- warp数据同步：
  - 软件插入同步标记
  - 硬件barrier执行同步功能
  - 未来考虑使用RVWMO完善同步模型

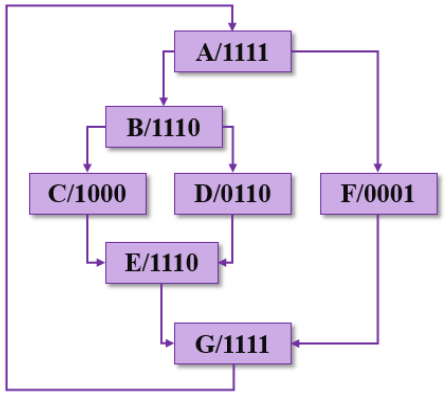
```
//warp0,1,2,3
vsoxei32.v
barrier
vloxei32.v
```



“承影” GPGPU微架构

- 灵活的硬件分支：使用自定义指令完成mask控制，同时保留RVV中使用通用向量寄存器v0作mask
  - 分支问题：同一warp内不同thread需执行不同路径
- 硬件调度发射：指令依赖判断和调度由硬件而非control code实现。后续会提供更多策略与指令配置选项
  - 多warp发射顺序：保证功能提升性能

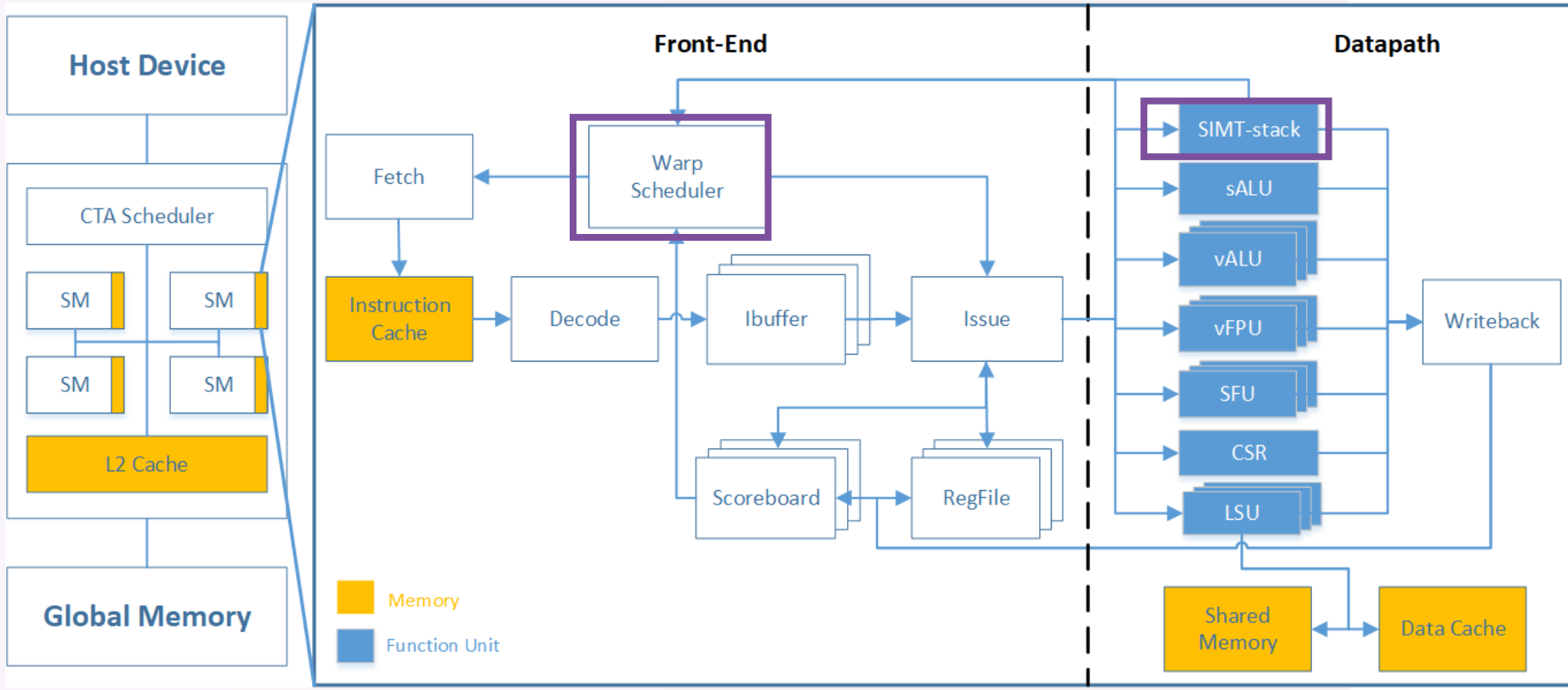




“承影” GPGPU微架构 - 分支处理

灵活硬件分支机制：

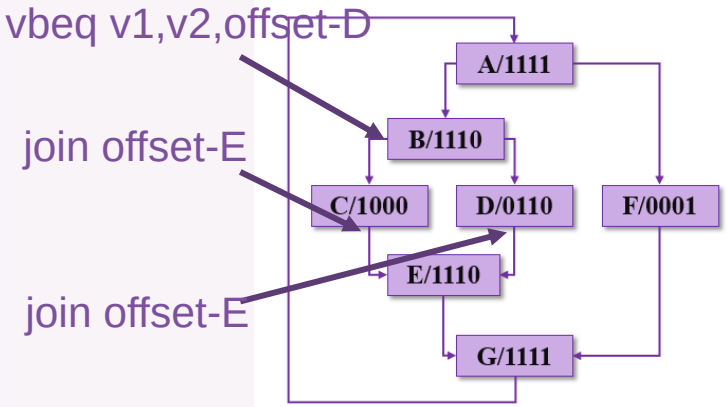
- SIMT-stack硬件管理
  - 不占用通用寄存器，减少流水线停顿，嵌套分支也能较快处理
  - 仅用三条指令完成一次完整分支-合并操作
- 保留对RVV mask的支持



“承影” GPGPU微架构 - 分支处理

灵活硬件分支机制：

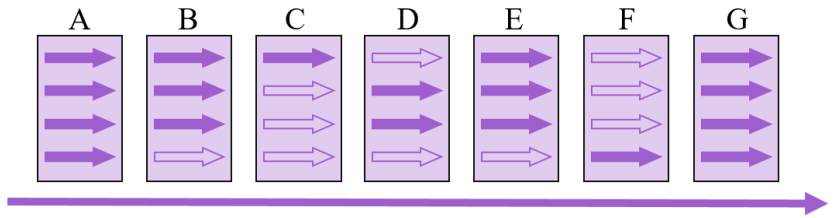
- SIMT-stack硬件管理
  - 不占用通用寄存器，减少流水线停顿，嵌套分支也能较快处理
  - 仅用三条指令完成一次完整分支-合并操作
- 保留对RVV mask的支持



(a) 程序样例

rPC	PC	掩码(mask)
-	G	1111

(h) 分支1执行完毕



rPC	PC	掩码(mask)
-	A	1111

(b) 栈初始状态

rPC	PC	掩码(mask)
-	G	1111
G	F	0001
G	B	1110

(c) 分支1发生后

rPC	PC	掩码(mask)
-	G	1111
G	F	0001
G	E	1110
E	D	0110
E	C	1000

(d) 分支2发生后

rPC	PC	掩码(mask)
-	G	1111
G	F	0001
G	E	1110
E	D	0110

(e) C段程序执行完毕

rPC	PC	掩码(mask)
-	G	1111
G	F	0001
G	E	1110

(f) 分支2执行完毕

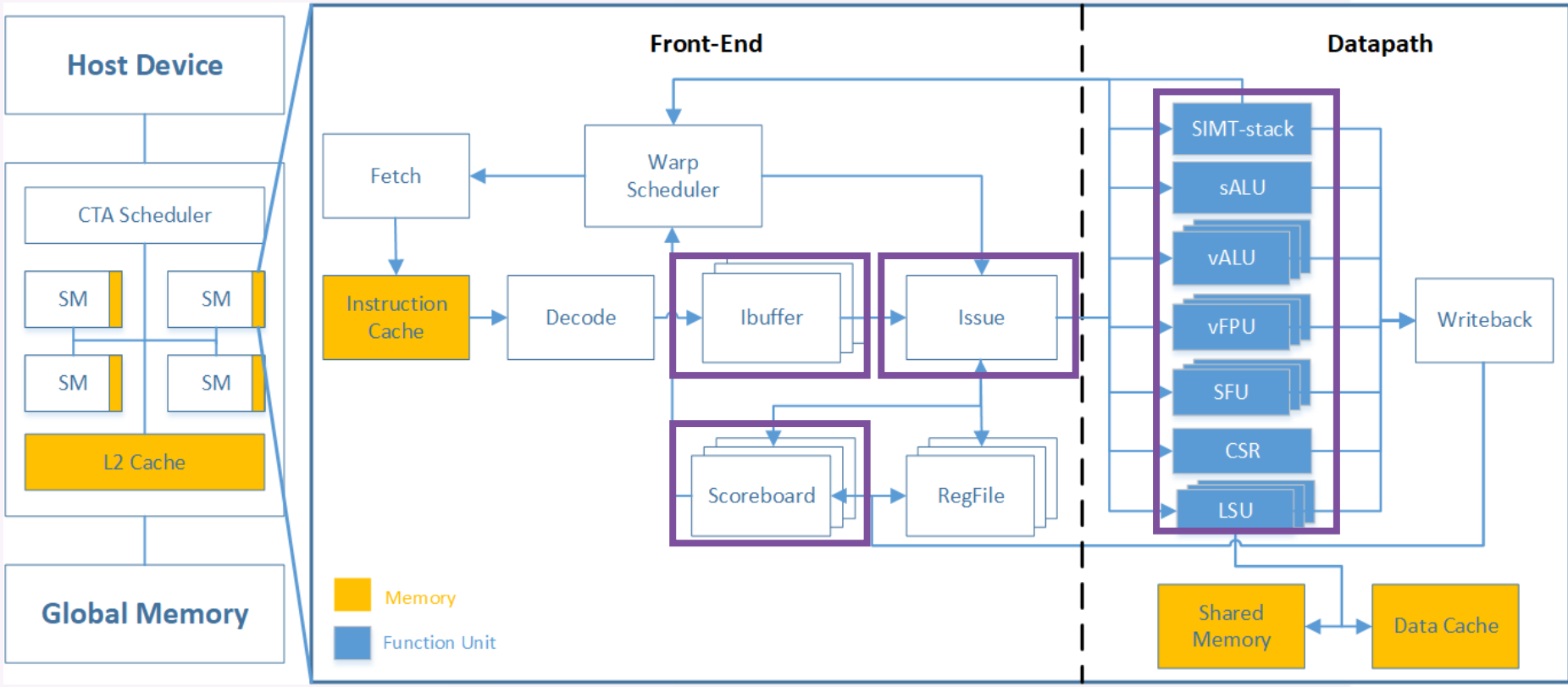
rPC	PC	掩码(mask)
-	G	1111
G	F	0001

(g) E段程序执行完毕

“承影” GPGPU微架构 - warp硬件调度

硬件调度发射执行：

- ibuffer暂存warp信息
- Scoreboard记录依赖
- Round-Robin发射
- 无需编译器参与，即具备一定调度效果，但易受特定case影响

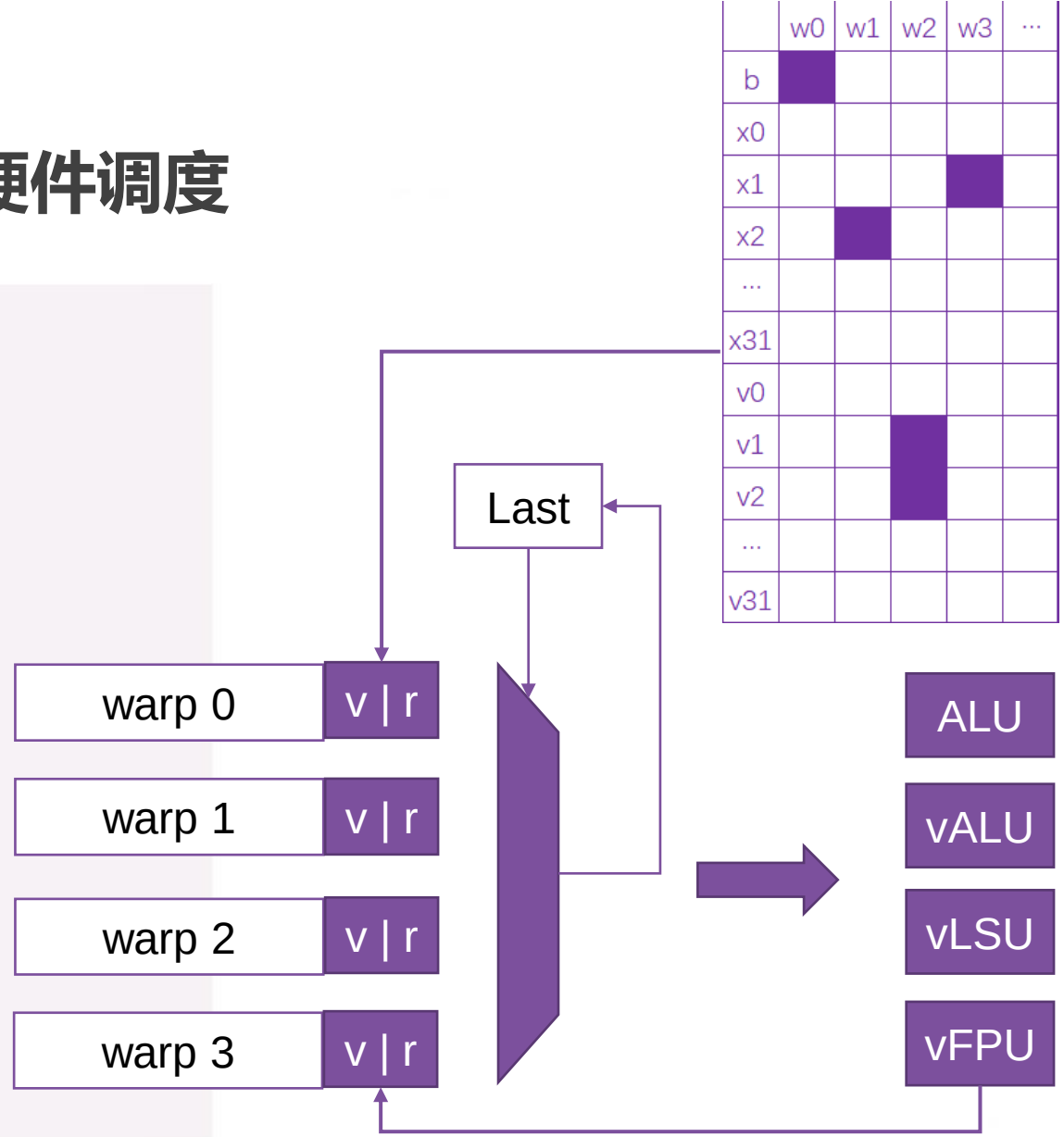


“承影” GPGPU微架构 - warp硬件调度

硬件调度发射执行：

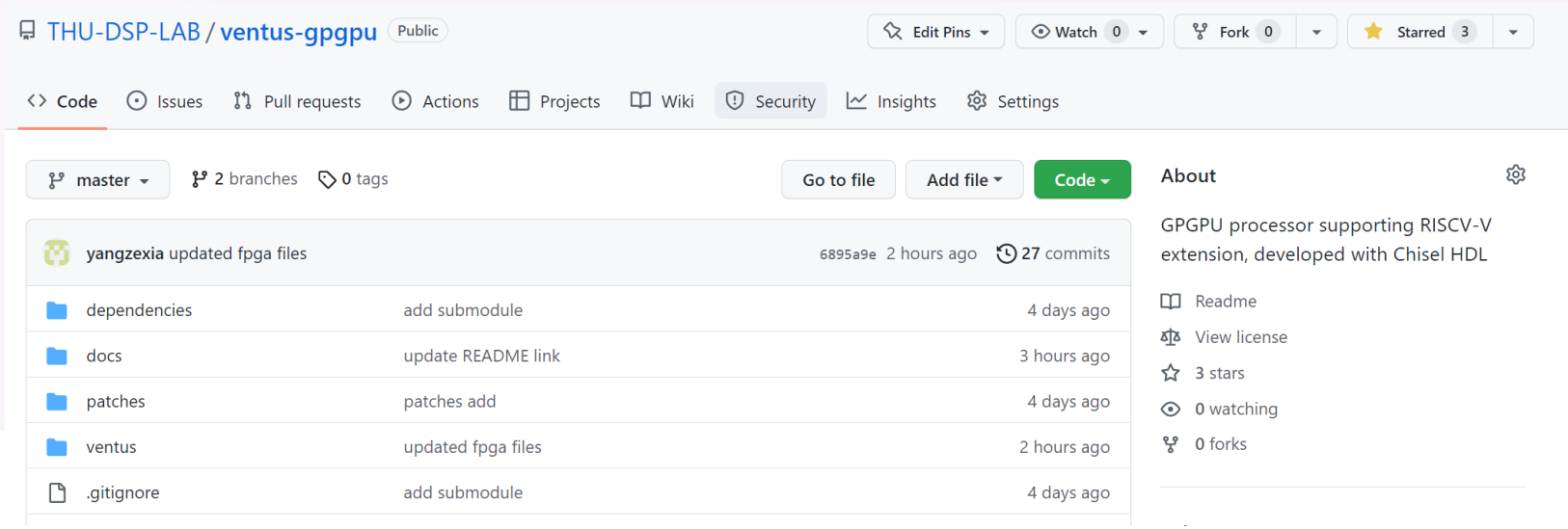
- ibuffer暂存warp信息
- Scoreboard记录依赖
- Round-Robin发射

- warp切换掩盖执行延迟
- 使功能单元负载充分
- 后续会设计更多配置方式



设计实现

- 使用Chisel HDL实现，利用Chisel高效参数化特点可生成不同规模硬件单元，应用无需更改仍能运行
- 代码开源在
  - https://github.com/THU-DSP-LAB/ventus-gpgpu
  - https://gitee.com/tsinghua-sh/ventus-gpgpu



设计实现

- 修改GCC工具链汇编器，并基于Spike开发了配套的ISA仿真器，可在指令精度下启动任务，运行带自定义指令的多warp汇编程序
- 完成了常见benchmark如saxpy，gaussian等汇编代码在RTL上的功能验证，并有与C程序和CUDA程序的运行结果对比

CUDA C++

```
__global__ void Fan1(float *m_cuda, float *a_cuda, int
Size, int t)
{
    if(threadIdx.x + blockIdx.x * blockDim.x >= Size-1-t)
return;
    *(m_cuda+Size*(blockDim.x*blockIdx.x+threadIdx.x+t+1)+
t) =
*(a_cuda+Size*(blockDim.x*blockIdx.x+threadIdx.x+t+1)+t) /
*(a_cuda+Size*t+t);
}
```

Our Assembly Code

```
Kernel1:
addi        x20, x7, 1              # t+1
vadd.vx     v20, v4, x20            # tid.y + (t+1)
vmv.v.x     v21, x8                 # size
vbge        v20, v21, K1_A
vbne        v3, v31, K1_END         # | tid.x==0 then exec
vmul.vx     v20, v20, x8            # | i.e. the first warp(s)
slli        x17, x7, 2              # [][t]
mul         x20, x17, x8            # [t][0]
add         x20, x20, x17           # [t][t]
slli        x16, x8, 2              # [1][0] i.e. stride
add         x15, x20, x16           # [t+1][t]
add         x15, x15, x10           # A[t+1][t]

vlse32.v    v10, (x15), x16         # A[(t+1)+tid.y][t]
add         x15, x20, x10
lw          x15, 0(x15)             # A[t][t]
vmv.v.x     v11, x15
vfdiv.vv    v10, v10, v11           # FP DIV

add         x15, x20, x16           # [t+1][t]
add         x15, x15, x9            # M[t+1][t]
vsse32.v    v10, (x15), x16         # save to M

K1_END:
join        x0, K1_A
K1_A:
join        x0, K1_B
K1_B:
barrier     x0, x0, x0               # syncthread
```

CUDA PTX

```
.visible .entry _Z4Fan1PfS_ii(
    .param .u64 _Z4Fan1PfS_ii_param_0,
    .param .u64 _Z4Fan1PfS_ii_param_1,
    .param .u32 _Z4Fan1PfS_ii_param_2,
    .param .u32 _Z4Fan1PfS_ii_param_3
)
{
    .reg .pred  %p<2>;
    .reg .f32   %f<4>;
    .reg .b32   %r<13>;
    .reg .b64   %rd<15>;
    ld.param.u64    %rd1, [_Z4Fan1PfS_ii_param_0];
    ld.param.u64    %rd2, [_Z4Fan1PfS_ii_param_1];
    ld.param.u32    %r2, [_Z4Fan1PfS_ii_param_2];
    ld.param.u32    %r3, [_Z4Fan1PfS_ii_param_3];
    mov.u32     %r4, %ntid.x;
    mov.u32     %r5, %ctaid.x;
    mov.u32     %r6, %tid.x;
    mad.lo.s32  %r1, %r5, %r4, %r6;
    not.b32     %r7, %r3;
    add.s32     %r8, %r7, %r2;
    setp.ge.u32     %p1, %r1, %r8;
    @%p1 bra    $L__BB0_2;

    cvta.to.global.u64  %rd3, %rd2;
    cvta.to.global.u64  %rd4, %rd1;
    add.s32     %r9, %r1, %r3;
    add.s32     %r10, %r9, 1;
    mul.lo.s32  %r11, %r10, %r2;
    cvt.u64.u32     %rd5, %r11;
    cvt.s64.s32     %rd6, %r3;
    add.s64     %rd7, %rd5, %rd6;
    shl.b64     %rd8, %rd7, 2;
    add.s64     %rd9, %rd3, %rd8;
    mul.lo.s32  %r12, %r3, %r2;
    cvt.s64.s32     %rd10, %r12;
    add.s64     %rd11, %rd10, %rd6;
    shl.b64     %rd12, %rd11, 2;
    add.s64     %rd13, %rd3, %rd12;
    ld.global.f32   %f1, [%rd13];
    ld.global.f32   %f2, [%rd9];
    div.rn.f32  %f3, %f2, %f1;
    add.s64     %rd14, %rd4, %rd8;
    st.global.f32   [%rd14], %f3;

$L__BB0_2:
    ret;
}
```

设计实现

- 在Xilinx VCU128上开发AXI驱动程序，使用MicroBlaze作为Host进行任务发射，用PL搭建“承影”GPGPU进行计算，通过DDR共享内存
- 40nm SMIC综合worst case下为350MHz
- Xilinx VCU128上按照4warp8thread配置可部署160个core，100MHz频率，理论峰值算力为32Gflops，可同时驻留1280个thread

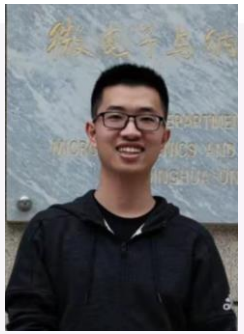
展望

- 后续工具链开发：
  - 调试和UVM验证工具
  - RVV自动向量化工具
  - CUDA – LLVM – RVV的路线
- 后续架构改进：
  - 借鉴向量处理器思路
  - 借鉴现有GPGPU架构，如dynamic parallelism
  - RVWMO与GPGPU的结合
  - 加入图形功能
  - 加入tensor core和transformer等执行单元

参与开源工作人员名单

何虎

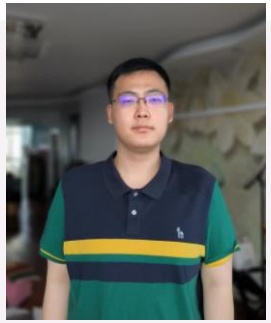
清华大学
集成电路学院



杨轲翔

金楚丰

李京洲

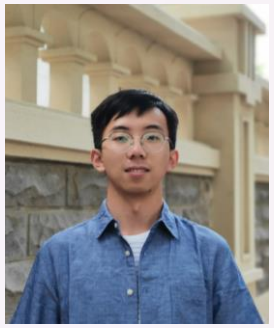


周重淳

于芳菲

刘旭东

杨泽夏



张泰然

郑名宸

邹昕航

邓仰东老师和马立伟博士亦对此项目做出贡献，在此一并感谢！

THANK YOU

上海清华国际创新中心
集成电路研究平台
International Innovation Center of Tsinghua University Shanghai
Integrated Circuit Research Platform