SPACEMIT
进迭时空

面向RISC-V同构融合CPU处理器的Triton算子编译器设计与实践

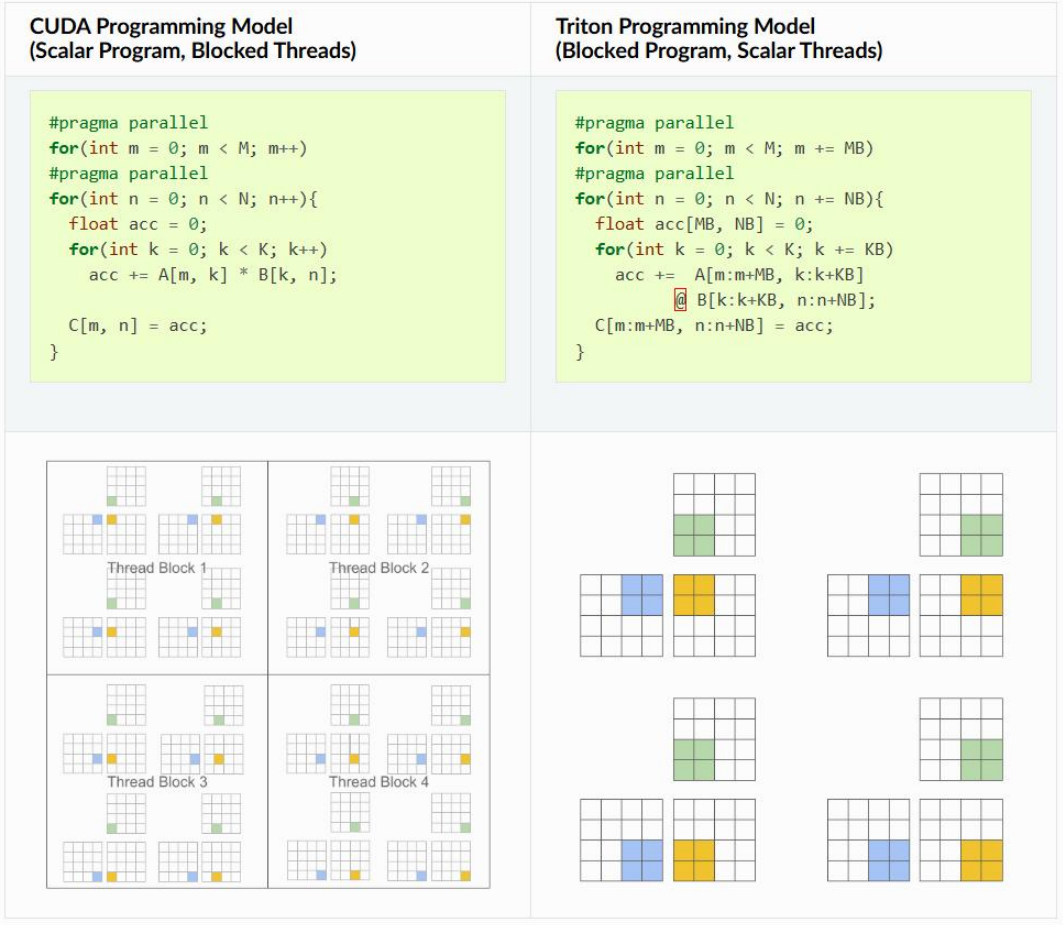
进迭时空 黄竞辉

CONTENTS

目录

Why Triton

- 编程模型收敛：多线程、Tile Base
- CUDA兼容性问题
- Python First or Pytorch First
- 新AI架构需要一个活跃可持续的DSL社区



RISCV Triton-CPU软件栈

- 统一内存，统一架构，消除Host-Device隔阂
- AICPU提供完整的语法支持
- 不止作为Pytorch加速后端，也直接提供Kernel API

PyTorch

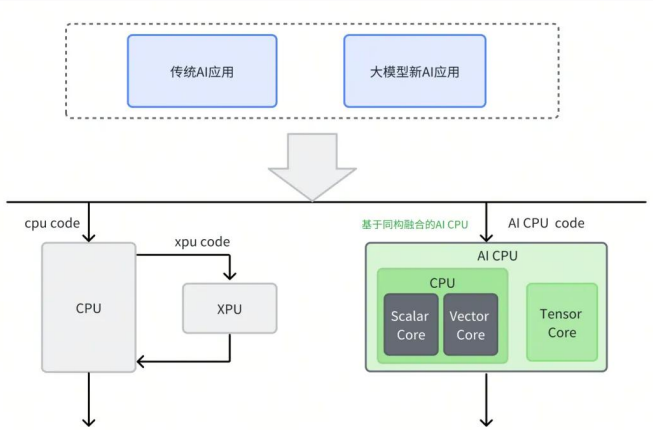
ONNX Runtime

...

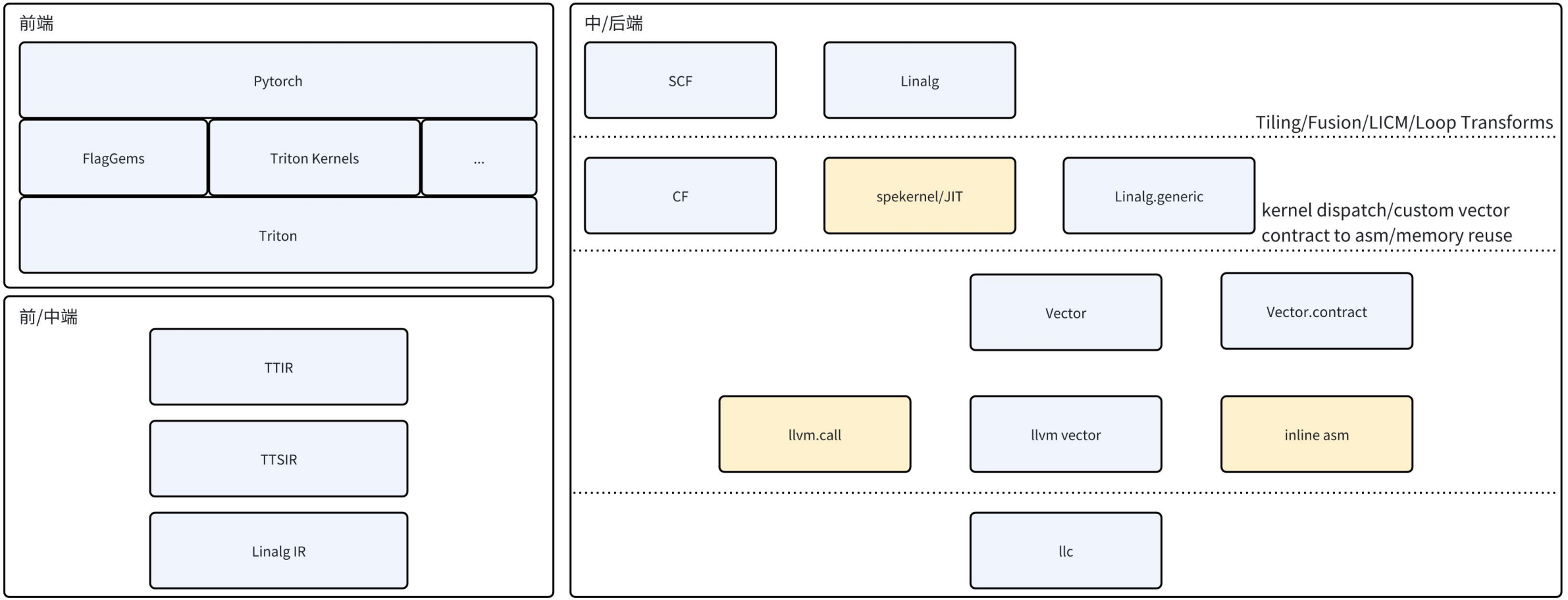
Triton

FlagOpen

SPACEMIT 进迭时空

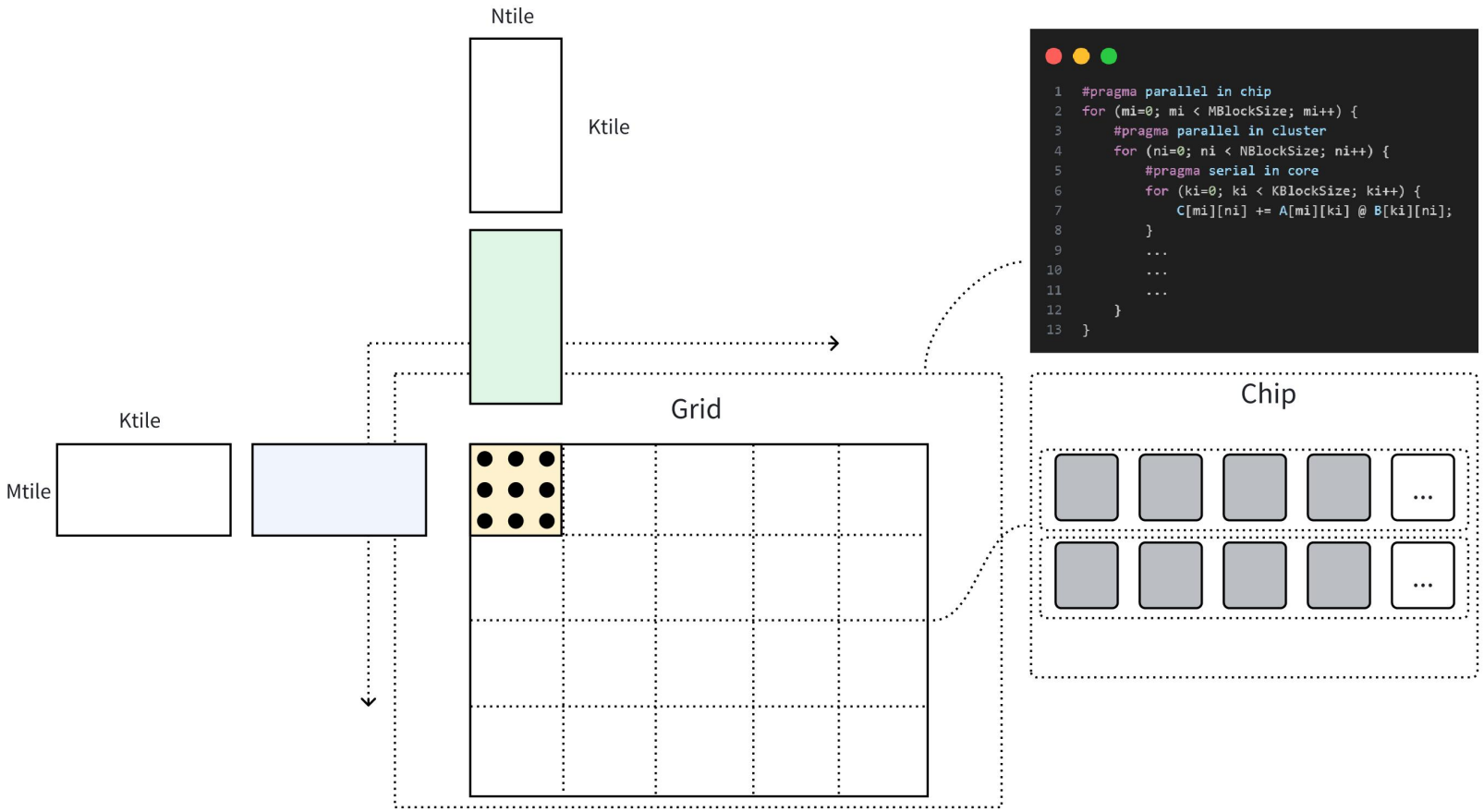


RISCV Triton-CPU软件栈



RISCV Triton-CPU实践-网格化

- 计算网格化
  - Grid ND->
    Die/Cluster/Core，提升局部性
- Cluster特化
  - Kernel编程视角视为单核
  - Share Memory
  - 核内向量化、Tensor化



RISCV Triton-CPU实践-块级访存

- 着重block_ptr的使用，更贴合CPU编程思路
- 保持Load with Mask的使用方式
- 以Cluster视角的多线程kernel与单线程一致

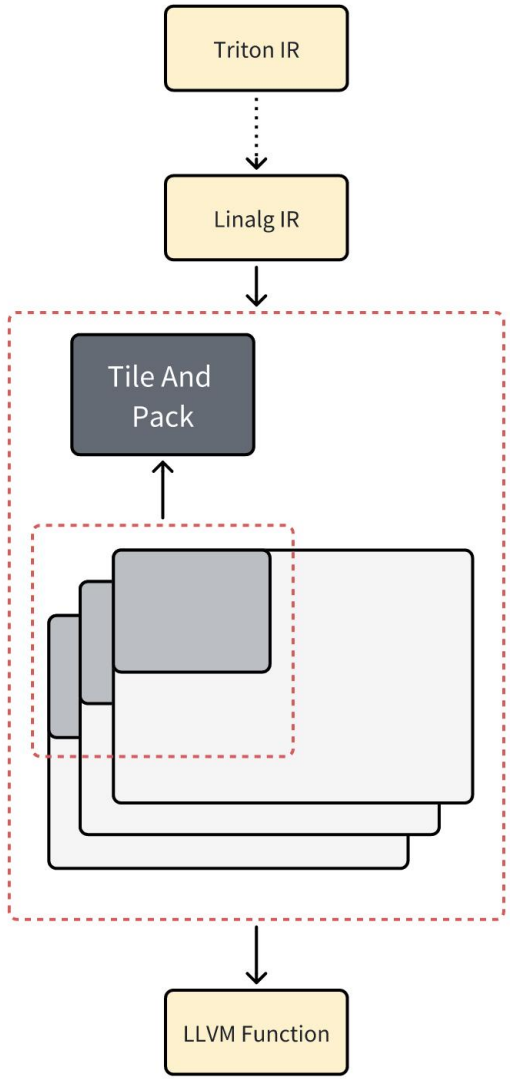
```
%6 = tt.make_tensor_ptr %arg0, [%3, %4], [%5, %c1_i64], [%2, %c0_i32] {order = array<i32: 1, 0>} :
%7 = arith.muli %1, %c16_i32 : i32 loc(#loc6)
%8 = arith.extsi %arg4 : i32 to i64 loc(#loc7)
%9 = arith.extsi %arg7 : i32 to i64 loc(#loc7)
%10 = tt.make_tensor_ptr %arg1, [%4, %8], [%9, %c1_i64], [%c0_i32, %7] {order = array<i32: 1, 0>} :
%11 = tt.load %6 {boundaryCheck = array<i32: 0, 1>} : !tt.ptr<tensor<16x1024xf32>> loc(#loc8)
%12 = tt.load %10 {boundaryCheck = array<i32: 0, 1>} : !tt.ptr<tensor<1024x16xf32>> loc(#loc9)
%13 = tt.dot %11, %12, %cst : tensor<16x1024xf32> * tensor<1024x16xf32> -> tensor<16x16xf32> loc(#l
%14 = arith.extsi %arg8 : i32 to i64 loc(#loc11)
%15 = tt.make_tensor_ptr %arg2, [%3, %8], [%14, %c1_i64], [%2, %7] {order = array<i32: 1, 0>} : <te
tt.store %15, %13 {boundaryCheck = array<i32: 0, 1>} : !tt.ptr<tensor<16x16xf32>> loc(#loc12)
```

```python
pid_m = tl.program_id(0)
pid_n = tl.program_id(1)

a_block_ptr = tl.make_block_ptr(
    base=a_ptr,
    shape=[M, K],
    strides=[stride_am, stride_ak],
    offsets=[pid_m * BLOCK_SIZE_M, 0],
    block_shape=[BLOCK_SIZE_M, BLOCK_SIZE_K],
    order=[1, 0],
)

b_block_ptr = tl.make_block_ptr(
    base=b_ptr,
    shape=[K, N],
    strides=[stride_bk, stride_bn],
    offsets=[0, pid_n * BLOCK_SIZE_N],
    block_shape=[BLOCK_SIZE_K, BLOCK_SIZE_N],
    order=[1, 0],
)

accumulator = tl.zeros((BLOCK_SIZE_M, BLOCK_SIZE_N), dtype=tl.float32)

if EVEN_K:
    a = tl.load(a_block_ptr, boundary_check=(0, 1))
    b = tl.load(b_block_ptr, boundary_check=(0, 1))
    accumulator += tl.dot(a, b, allow_tf32=False)
else:
    for k in range(0, tl.cdiv(K, BLOCK_SIZE_K)):
        a = tl.load(a_block_ptr, boundary_check=(0, 1))
        b = tl.load(b_block_ptr, boundary_check=(0, 1))
        accumulator += tl.dot(a, b, allow_tf32=False)
        a_block_ptr = tl.advance(a_block_ptr, (0, BLOCK_SIZE_K))
        b_block_ptr = tl.advance(b_block_ptr, (BLOCK_SIZE_K, 0))
```

RISCV Triton-CPU实践-Gemm

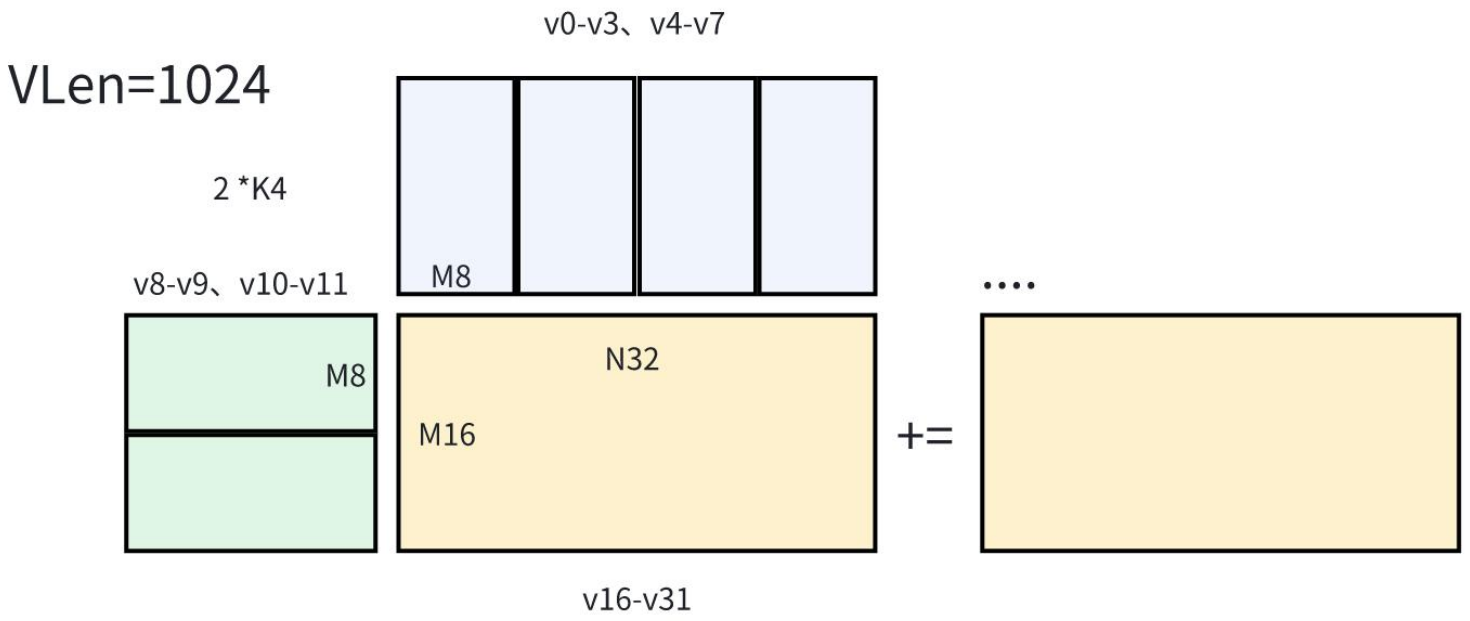
- spekernel对齐linalg.pack/mmt4d/...语义
- mmt4d->ime vmadot/vector load store
- pack->vector

```
%subview_14 = memref.subview %reinterpret_cast_6[0, 0] [%77, %83] [1, 1] : memref<?x?xf32, stri
%subview_15 = memref.subview %reinterpret_cast_7[0, 0] [%77, %84] [1, 1] : memref<?x?xf32, stri
%subview_16 = memref.subview %alloc_4[0, 0] [%77, %83] [1, 1] : memref<32x32xf32> to memref<?x?
%subview_17 = memref.subview %alloc_4[0, %83] [%77, %84] [1, 1] : memref<32x32xf32> to memref<?
memref.copy %subview_14, %subview_16 : memref<?x?xf32, strided<[?, ?], offset: ?>> to memref<?x
memref.copy %subview_15, %subview_17 : memref<?x?xf32, strided<[?, ?], offset: ?>> to memref<?x
spekernel.generic "pack" ins(%alloc_3 : memref<64x32xf32>) outs(%alloc_0 : memref<16x4x4x8xf32>
spekernel.generic "pack" ins(%alloc_4 : memref<32x32xf32>) outs(%alloc_1 : memref<2x4x16x8xf32>
spekernel.generic "mmt4d" ins(%alloc_0, %alloc_1 : memref<16x4x4x8xf32>, memref<2x4x16x8xf32>)
%85 = arith.addi %arg16, %c32 : index
%86 = arith.addi %arg17, %34 : index
scf.yield %85, %86 : index, index
```



RISCV Triton-CPU实践-Gemm

- Gemm Analysis
  - 寄存器用满，M/N First->K
  - Core上缓存用满，K First->M/N



RISCV Triton-CPU实践-Gemm Lower

- Gemm Codegen

```
%3 = scf.for %arg3 = %c0 to %c512 step %c128 iter_args(%arg4 = %arg2) -> (tensor<256x512xf32>) {
  %extracted_slice = tensor.extract_slice %arg1[0, %arg3] [256, 128] [1, 1] : tensor<256x512xf32> to tensor<256x128xf32>
  %extracted_slice_0 = tensor.extract_slice %arg4[0, %arg3] [256, 128] [1, 1] : tensor<256x512xf32> to tensor<256x128xf32>
  %4 = spekernel.generic "pack" ins(%extracted_slice : tensor<256x128xf32>) outs(%1 : tensor<4x8x32x32xf32>) (%c256, %c128, %c4,
  %5 = scf.for %arg5 = %c0 to %c256 step %c16 iter_args(%arg6 = %extracted_slice_0) -> (tensor<256x128xf32>) {
    %extracted_slice_1 = tensor.extract_slice %arg0[%arg5, 0] [16, 256] [1, 1] : tensor<256x256xf32> to tensor<16x256xf32>
    %extracted_slice_2 = tensor.extract_slice %arg6[%arg5, 0] [16, 128] [1, 1] : tensor<256x128xf32> to tensor<16x128xf32>
    %6 = spekernel.generic "pack" ins(%extracted_slice_1 : tensor<16x256xf32>) outs(%0 : tensor<1x8x16x32xf32>) (%c16, %c256, %c1
    %7 = spekernel.generic "pack" ins(%extracted_slice_2 : tensor<16x128xf32>) outs(%2 : tensor<1x4x16x32xf32>) (%c16, %c128, %c1
    %8 = spekernel.generic "mmt4d" ins(%6, %4 : tensor<1x8x16x32xf32>, tensor<4x8x32x32xf32>) outs(%7 : tensor<1x4x16x32xf32>) (%
    %9 = spekernel.generic "unpack" ins(%8 : tensor<1x4x16x32xf32>) outs(%extracted_slice_2 : tensor<16x128xf32>) (%c1, %c4, %c16
    %inserted_slice_3 = tensor.insert_slice %9 into %arg6[%arg5, 0] [16, 128] [1, 1] : tensor<16x128xf32> into tensor<256x128xf32
    scf.yield %inserted_slice_3 : tensor<256x128xf32>
  }
  %inserted_slice = tensor.insert_slice %5 into %arg4[0, %arg3] [256, 128] [1, 1] : tensor<256x128xf32> into tensor<256x512xf32>
  scf.yield %inserted_slice : tensor<256x512xf32>
}
return %3 : tensor<256x512xf32>
}
```

RISCV Triton-CPU实践-Gemm

- Gemm Codegen

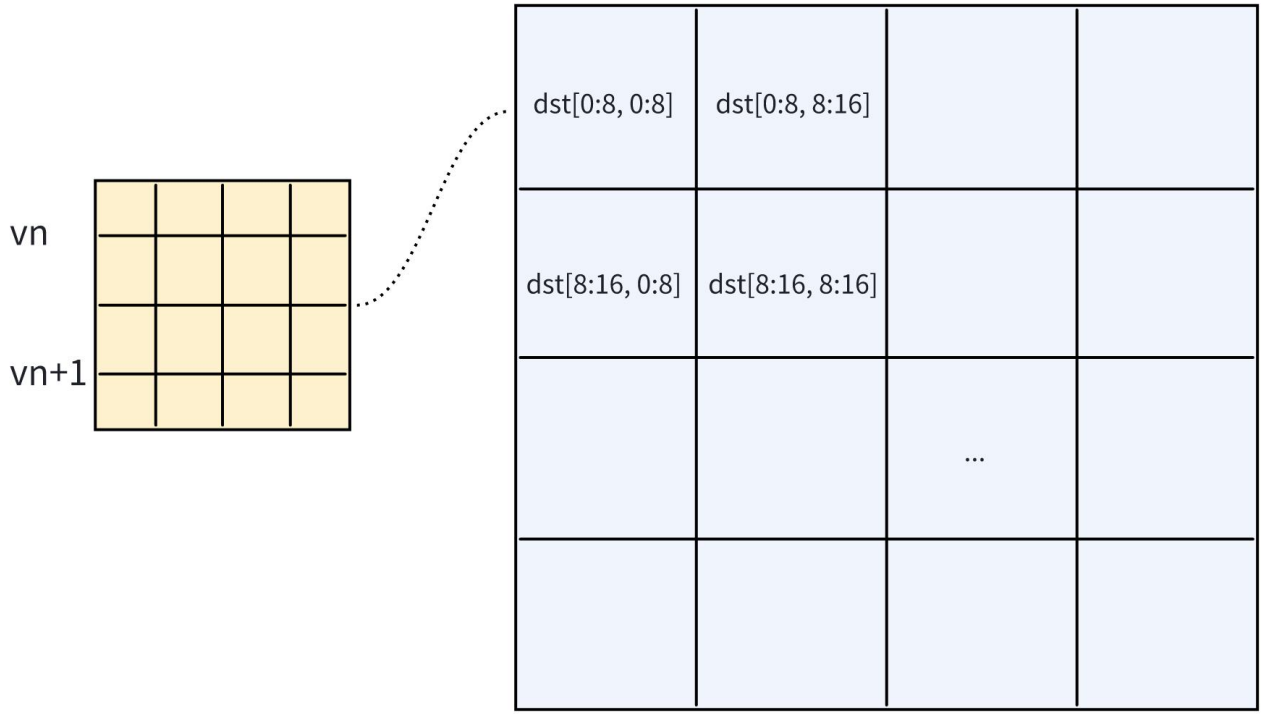
```
// load b
%0 = vector.load [...] : memref<?x?x32x4xf32>, vector<4x32xf32>
// load a
%1 = vector.load [...] : memref<?x?x16x4xf32>, vector<2x32xf32>
// vfmadot -> 2x8x4 @ 4x8x4 => 2x4x8x8
%2 = vector.contract {...} %1, %0 : vector<2x32xf32>, vector<4x32xf32> into vector<16x32xf32>

```

```
// pack A -> [m_block_size, k_block_size, mb, kb]
// pack B -> [n_block_size, k_block_size, nb, kb]
// packA @ packC -> [m_block_size, k_block_size, mb, nb]
for (mi : m_block_size) {
    for (ni : n_block_size) {
        // cf loop
        for (ki : k_block_size) {
            // vector.contract
            C[mi, ni, :, :] += A[mi, ki, :, :] @ B[ni, ki, :, :]
        }
    }
}
```

```
vle32.v ...
vle32.v ...
...
vmadot $(acc:0), $(lhs:0), $(rhs:0)
vmadot $(acc:2), $(lhs:0), $(rhs:1)
...
vmadot $(acc:8), $(lhs:1), $(rhs:0)
vmadot $(acc:10), $(lhs:1), $(rhs:1)
...
```

RISCV Triton-CPU实践-Gemm

- Gemm Codegen



RISCV Triton-CPU实践-Conv

- Conv Codegen

```
// kernel_shape=[3,3]
// %acc: vector<16x32xf32>
// load b
%0 = vector.load [...] : memref<?x?x32x4xf32>, vector<9x32xf32>
// load a h0
%1 = vector.load [...] : memref<?x?x16x4xf32>, vector<3x32xf32>
// vfmadotn -> 3x8x4 @ 9x8x4 => 2x4x8x8
%2 = vector.contract {...} %1, %0 %acc : vector<3x32xf32>, vector<9x32xf32> into vector<16x32xf32>
// load a h1
%3 = vector.load [...] : memref<?x?x16x4xf32>, vector<3x32xf32>
// vfmadotn -> 3x8x4 @ 9x8x4 => 2x4x8x8
%4 = vector.contract {...} %3, %0 %2 : vector<3x32xf32>, vector<9x32xf32> into vector<16x32xf32>
// load a h2
%5 = vector.load [...] : memref<?x?x16x4xf32>, vector<3x32xf32>
// vfmadotn -> 3x8x4 @ 9x8x4 => 2x4x8x8
%6 = vector.contract {...} %5, %0 %4 : vector<3x32xf32>, vector<9x32xf32> into vector<16x32xf32>

```

```
vle32.v ...
vle32.v ...
...
vmadot $(acc:0), $(lhs:0), $(rhs:0)
vmadot1 $(acc:0), $(lhs:0), $(rhs:0)
vmadot2 $(acc:0), $(lhs:0), $(rhs:0)
...
vmadot $(acc:2), $(lhs:0), $(rhs:1)
vmadot1 $(acc:2), $(lhs:0), $(rhs:1)
vmadot2 $(acc:2), $(lhs:0), $(rhs:1)
...
vse32.v
vse32.v
```

RoadMap

开源

- Triton前中端以及后端Compiler逐步开源

更彻底的Codegen

- 矩阵计算序列与Vector融合调优
- AME
- triton inline asm

分布式支持

- 关注到triton-distribute，分布式通信原语及硬件支持

SPACEMIT
进迭时空

Thanks

以RISC-V架构数智未来

RISC-V ARCHITECTURE FOR INTELLIGENT FUTURE

进迭时空（杭州）科技有限公司
SPACEMIT (Hangzhou) Technology Co., Ltd

电话：0571-89000775
网址：www.spacemit.com
地址：浙江省杭州市余杭区五常街道未来天空中心b座701