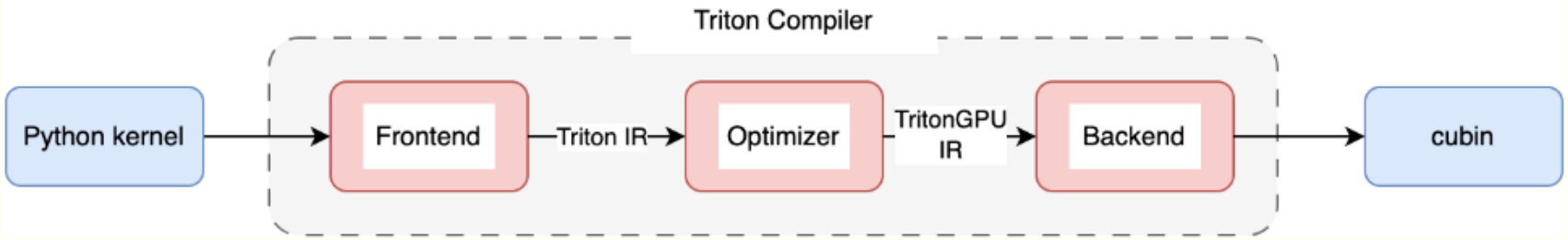
Triton编译流程

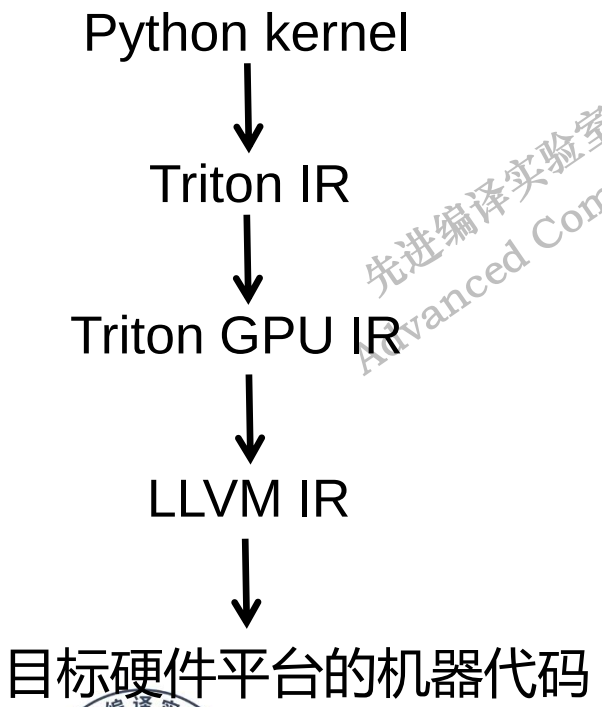
嘉宾：董贞汝

Triton编译器架构



- **前端（Frontend）：** 用于将用户使用Python编写的kernel或者Pytorch2.0中通过TorchInductor生成的TritonKernel转换为对应的Triton IR
- **优化器（Optimizer）：** 通过各类pass将Triton IR逐步转换并优化为TritonGPU IR
- **后端（Backend）：** 将TritonGPU IR逐步转换为LLVM IR

Triton 编译流程



先进编译实验室
Advanced Compiler

Triton IR

- 高级抽象
- 操作表示
- 优化
- 转换为Triton GPU IR

add_kernel.ttir如下所示：

```
module {
  tt.func public @add_kernel_0d1d2d3de(%arg0: !tt.ptr<f32, 1> {tt.divisibility = 16 : i32}, %arg1: !tt.ptr<f32, 1> {tt.divisibility = 16 : i32}, %arg2: !tt.ptr<f32, 1>
{tt.divisibility = 16 : i32}, %arg3: i32 {tt.divisibility = 16 : i32, tt.max_divisibility = 16 : i32}) attributes {noinline = false} {
    %c1024_i32 = arith.constant 1024 : i32
    %0 = tt.get_program_id x : i32
    %1 = arith.muli %0, %c1024_i32 : i32
    %2 = tt.make_range {end = 1024 : i32, start = 0 : i32} : tensor<1024xi32>
    %3 = tt.splat %1 : (i32) -> tensor<1024xi32>
    %4 = arith.addi %3, %2 : tensor<1024xi32>
    %5 = tt.splat %arg3 : (i32) -> tensor<1024xi32>
    %6 = arith.cmpi slt, %4, %5 : tensor<1024xi32>
    %7 = tt.splat %arg0 : (!tt.ptr<f32, 1>) -> tensor<1024x!tt.ptr<f32, 1>>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32, 1>>, tensor<1024xi32>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
    %10 = tt.splat %arg1 : (!tt.ptr<f32, 1>) -> tensor<1024x!tt.ptr<f32, 1>>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32, 1>>, tensor<1024xi32>
    %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
    %13 = arith.addf %9, %12 : tensor<1024xf32>
    %14 = tt.splat %arg2 : (!tt.ptr<f32, 1>) -> tensor<1024x!tt.ptr<f32, 1>>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32, 1>>, tensor<1024xi32>
    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
    tt.return
  }
}
```

Triton GPU IR

- 硬件特定优化
- 并行性表示
- 性能优化
- 转换为LLVM IR

add_kernel.ttgir如下所示：

```
#blocked = #triton_gpu.blocked<{sizePerThread = [4], threadsPerWarp = [32], warpsPerCTA = [4], order = [0], CTAsPerCGA = [1], CTASplitNum = [1], CTAOrder = [0]}>
module attributes {"triton_gpu.compute-capability" = 86 : i32, "triton_gpu.num-ctas" = 1 : i32, "triton_gpu.num-warps" = 4 : i32, "triton_gpu.threads-per-warp" = 32 : i
32} {
  tt.func public @add_kernel_0d1d2d3de(%arg0: !tt.ptr<f32, 1> {tt.divisibility = 16 : i32}, %arg1: !tt.ptr<f32, 1> {tt.divisibility = 16 : i32}, %arg2: !tt.ptr<f32, 1>
{tt.divisibility = 16 : i32}, %arg3: i32 {tt.divisibility = 16 : i32, tt.max_divisibility = 16 : i32}) attributes {noinline = false} {
    %c1024_i32 = arith.constant 1024 : i32
    %0 = tt.get_program_id x : i32
    %1 = arith.muli %0, %c1024_i32 : i32
    %2 = tt.make_range {end = 1024 : i32, start = 0 : i32} : tensor<1024xi32, #blocked>
    %3 = tt.splat %1 : (i32) -> tensor<1024xi32, #blocked>
    %4 = arith.addi %3, %2 : tensor<1024xi32, #blocked>
    %5 = tt.splat %arg3 : (i32) -> tensor<1024xi32, #blocked>
    %6 = arith.cmpi slt, %4, %5 : tensor<1024xi32, #blocked>
    %7 = tt.splat %arg0 : (!tt.ptr<f32, 1>) -> tensor<1024x!tt.ptr<f32, 1>, #blocked>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32, 1>, #blocked>, tensor<1024xi32, #blocked>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32, #blocked>
    %10 = tt.splat %arg1 : (!tt.ptr<f32, 1>) -> tensor<1024x!tt.ptr<f32, 1>, #blocked>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32, 1>, #blocked>, tensor<1024xi32, #blocked>
    %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32, #blocked>
    %13 = arith.addf %9, %12 : tensor<1024xf32, #blocked>
    %14 = tt.splat %arg2 : (!tt.ptr<f32, 1>) -> tensor<1024x!tt.ptr<f32, 1>, #blocked>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32, 1>, #blocked>, tensor<1024xi32, #blocked>
    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32, #blocked>
    tt.return
  }
}
```

LLVM IR

- 平台无关
- 优化
- 代码生成
- 模块化

add_kernel.llir如下所示：

```
; ModuleID = 'LLVMDialectModule'
source_filename = "LLVMDialectModule"

define void @add_kernel_0d1d2d3de(ptr addrspace(1) %0, ptr addrspace(1) %1, ptr addrspace(1) %2, i32 %3) local_unnamed_addr !dbg !5 {
  %5 = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x(), !dbg !8
  %6 = shl i32 %5, 2, !dbg !8
  %7 = and i32 %6, 508, !dbg !8
  %8 = tail call i32 asm "mov.u32 $0, %ctaid.x;", "=r"() #1, !dbg !9
  %9 = shl i32 %8, 10, !dbg !10
  %10 = or i32 %9, %7, !dbg !11
  %11 = or i32 %10, 512, !dbg !11
  %12 = icmp slt i32 %10, %3, !dbg !12
  %13 = icmp slt i32 %11, %3, !dbg !12
  %14 = sext i32 %10 to i64, !dbg !13
  %15 = getelementptr float, ptr addrspace(1) %0, i64 %14, !dbg !13
  %16 = sext i32 %11 to i64, !dbg !13
  %17 = getelementptr float, ptr addrspace(1) %0, i64 %16, !dbg !13
  %18 = tail call { i32, i32, i32, i32 } asm sideeffect "mov.u32 $0, 0x0;\0A\09mov.u32 $1, 0x0;\0A\09mov.u32 $2, 0x0;\0A\09mov.u32 $3, 0x0;\0A\09@$5 ld.global.v4.b32 {
$0, $1, $2, $3 }, [ $4 + 0 ];", "=r,=r,=r,=r,l,b"(ptr addrspace(1) %15, i1 %12) #1, !dbg !14
  %19 = extractvalue { i32, i32, i32, i32 } %18, 0, !dbg !14
  %20 = extractvalue { i32, i32, i32, i32 } %18, 1, !dbg !14
  %21 = extractvalue { i32, i32, i32, i32 } %18, 2, !dbg !14
  %22 = extractvalue { i32, i32, i32, i32 } %18, 3, !dbg !14
  %23 = bitcast i32 %19 to float, !dbg !14
  %24 = bitcast i32 %20 to float, !dbg !14
  %25 = bitcast i32 %21 to float, !dbg !14
  %26 = bitcast i32 %22 to float, !dbg !14
  %27 = tail call { i32, i32, i32, i32 } asm sideeffect "mov.u32 $0, 0x0;\0A\09mov.u32 $1, 0x0;\0A\09mov.u32 $2, 0x0;\0A\09mov.u32 $3, 0x0;\0A\09@$5 ld.global.v4.b32 {
$0, $1, $2, $3 }, [ $4 + 0 ];", "=r,=r,=r,=r,l,b"(ptr addrspace(1) %17, i1 %13) #1, !dbg !14
```

Pytorch通过Inductor后端生成TritonKernel

示例一：

```
import torch
def fn(x):
   a = torch.cos(x)
   b = torch.sin(a)
   return b
new_fn = torch.compile(fn, backend="inductor")
input_tensor = torch.randn(10000).to(device="cuda:0")
a = new_fn(input_tensor)
```

示例二：

```
import torch
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', pretrained=True)
opt_model = torch.compile(model, backend="inductor")
opt_model(torch.randn(1,3,64,64))
```

运行命令：

(1) TORCH_COMPILE_DEBUG=1 python example.py

(2) TORCH_COMPILE_DEBUG=1 python resnet50.py

AdvancedCompiler
Tel: 13839830713