Triton-IR剖析（上）

嘉宾：董贞汝

Triton kernel到ttir转换过程

以01-vector_add.py为例简单了解转换过程

调用Triton Kernel（python/tutorials/01-vector-add.py）： add(x, y)

⇩

JIT调用编译器(triton/runtime/jit.py)： kernel = self.compile()

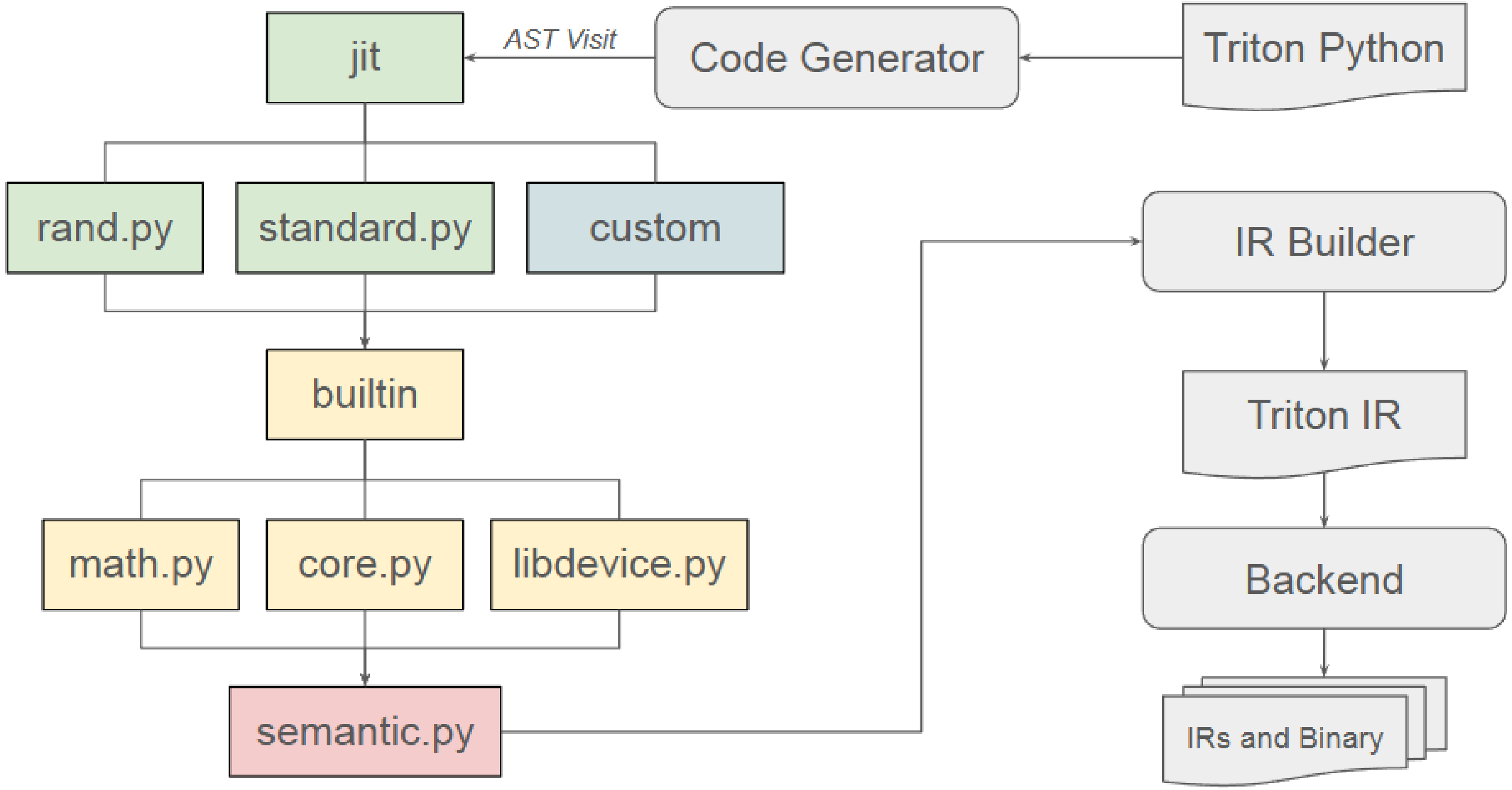
⇩

调用make_ir(triton/compiler/compiler.py)： src.make_ir()

⇩

调用ast去生成ttir(triton/compiler/compiler.py)： ast_to_ttir()，逻辑位于triton/compiler/code_generator.py文件中，具体地进行codeGen的遍历完成Triton Python到Triton IR（ttir）的转换

Triton kernel到ttir转换过程



- triton/language/random.py
  随机数生成
- triton/language/standard.py
  一些算子的标准实现
- triton/language/math.py
  数学算子的实现
- triton/language/core.py
  Triton核心操作的实现
- triton/language/extra/cuda/libdevice.py
  硬件相关的操作实现
- triton/language/semantic.py
  语法

先进编译实验室
Advanced Compiler

kernel与Triton IR的对应

先进编译实验室
Advanced Compiler

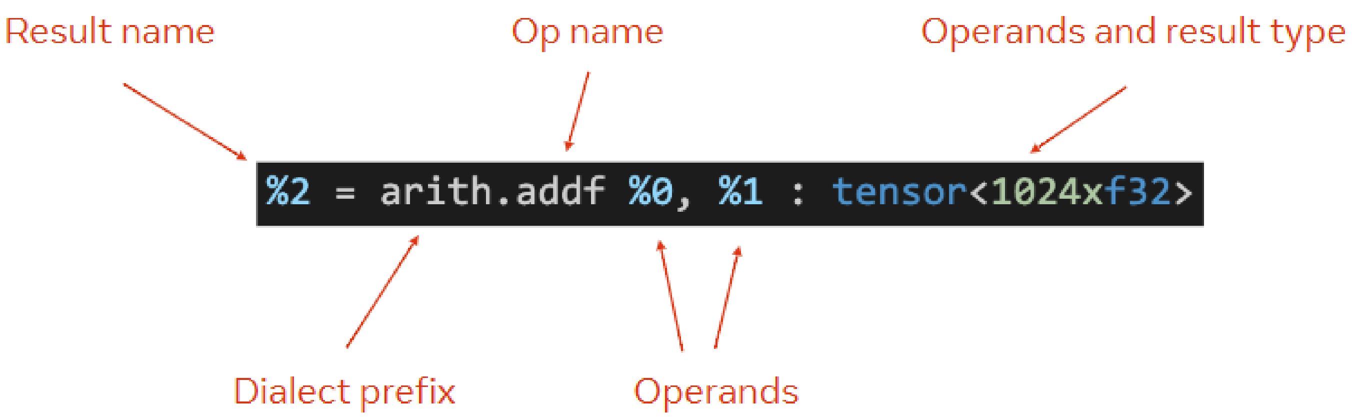
```python
import triton
import triton.language as tl

@triton.jit
def add_kernel(
    x_ptr,  # *Pointer* to first input vector.
    y_ptr,  # *Pointer* to second input vector.
    output_ptr,  # *Pointer* to output vector.
    n_elements,  # Size of the vector.
    BLOCK_SIZE: tl.constexpr,  # Number of elements each program should process.
                 # NOTE: `constexpr` so it can be used as a shape value.
):
    # There are multiple 'programs' processing different data. We identify which program
    # we are here:
    pid = tl.program_id(axis=0)  # We use a 1D launch grid so axis is 0.
    # This program will process inputs that are offset from the initial data.
    # For instance, if you had a vector of length 256 and block_size of 64, the programs
    # would each access the elements [0:64, 64:128, 128:192, 192:256].
    # Note that offsets is a list of pointers:
    block_start = pid * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    # Create a mask to guard memory operations against out-of-bounds accesses.
    mask = offsets < n_elements
    # Load x and y from DRAM, masking out any extra elements in case the input is not a
    # multiple of the block size.
    x = tl.load(x_ptr + offsets, mask=mask)
    y = tl.load(y_ptr + offsets, mask=mask)
    output = x + y
    # Write x + y back to DRAM.
    tl.store(output_ptr + offsets, output, mask=mask)
```

```mlir
module {
  tt.func public @add_kernel_01234(%arg0: !tt.ptr<f32>, %arg1: !tt.ptr<f32>, %arg2: !tt.ptr<f32>, %arg3: i32) {
    %c1024_i32 = arith.constant 1024 : i32
    %0 = tt.get_program_id {axis = 0 : i32} : 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>) -> tensor<1024x!tt.ptr<f32>>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
    %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, 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>) -> tensor<1024x!tt.ptr<f32>>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
    tt.return
  }
}
```

先进编译实验室
Advanced Compiler

MLIR的语法结构

IR Structure at a Glance



- MLIR uses the notion of operations
- Operations are “opaque functions” to MLIR
- IR is represented in SSA form
- Refer to these slides for more comprehensive examples

MLIR的语法结构

IR Structure at a Glance

- Dialects
  - Collection of operations, types, attributes, related transformation passes, analysis, etc.
  - Multiple dialects can co-exist within one module
  - E.g.: arith, math, func, llvm, memref, etc.
- Types
  - Supports both builtin and custom Dialect types
  - E.g.: i8, f32, memref, vector, tensor, etc.
- Attributes
  - In-place constant data on operations
  - E.g., comparison predicate, if perform boundary check on triton load, etc.

Triton-IR剖析（下）

嘉宾：董贞汝

先进编译实验室
Advanced Compiler

Triton IR中的优化pass

Triton Kernel → Triton Python Frontend → Triton Dialect (ttir) → Device Specific Lowering

TritonTypes.td:
- Float types
- Integer types
- Tensor types
- Pointer types
- ...

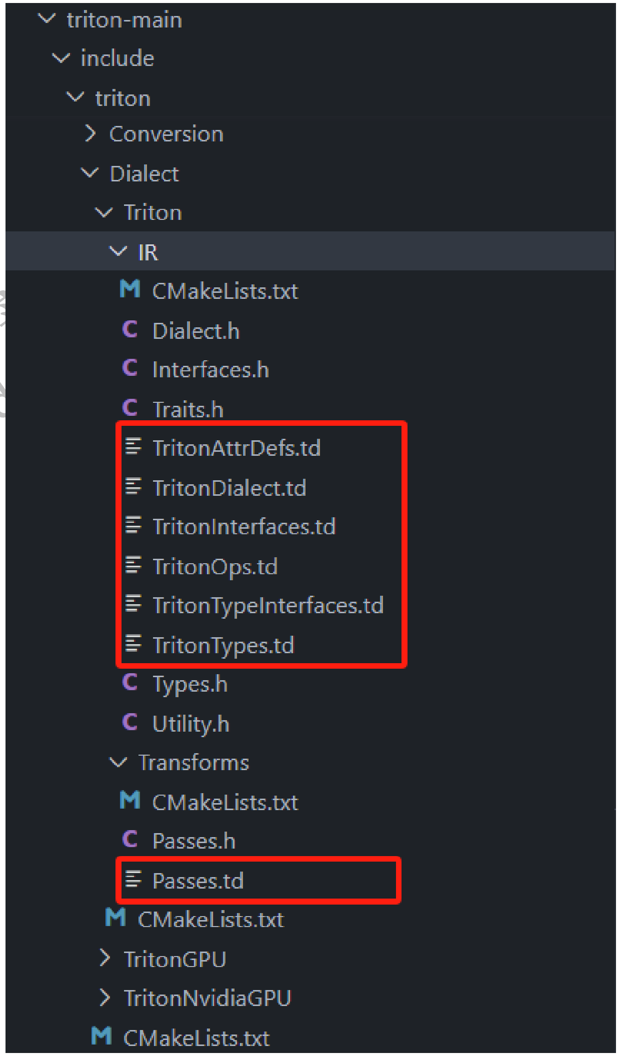
TritonOps.td:
- tt.addptr
- tt.load
- tt.store
- tt.dot
- ...

Passes.td:
- ...

TritonTypeInterfaces.td

TritonInterfaces.td

TritonAttrDefs.td
- Eviction policies
- Padding options
- Atomics attrs
- ...



先进编译实验室
Advanced Compiler

Triton IR中的优化pass

在 python/triton/compiler/compiler.py调用make_ttir去运行一些优化Pass。

```
for ext, compile_ir in list(stages.items())[first_stage:]:
    next_module = compile_ir(module, metadata)
```

NVIDIA GPU对应的make_ttir优化pass pipeline位于triton/backends/nvidia/compiler.py中

```
@staticmethod
def make_ttir(mod, metadata, opt):
    breakpoint()
    pm = ir.pass_manager(mod.context)
    pm.enable_debug()
    passes.common.add_inliner(pm)
    passes.ttir.add_rewrite_tensor_pointer(pm)
    passes.ttir.add_combine(pm)
    passes.common.add_canonicalizer(pm)
    passes.ttir.add_reorder_broadcast(pm)
    passes.common.add_cse(pm)
    passes.common.add_licm(pm)
    passes.common.add_symbol_dce(pm)
    pm.run(mod)
    return mod
```

Triton IR中的优化pass

Cse Pass: 公共子表达式消除（CSE, Common Subexpression Elimination）

功能：识别和消除重复计算的表达式，优化程序的运行时间和空间。

执行命令：triton-opt cse_before.ttir -cse &> cse_after.ttir

```
module {
  tt.func public @add_kernel_0d1d2d3d(%arg0: !tt.ptr<f32> {tt.divisibility = 16 : i32
    %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>) -> tensor<1024x!tt.ptr<f32>>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tens
    %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : te
    %13 = arith.addf %9, %12 : tensor<1024xf32>
    %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %16 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
    tt.return
  }
}
```

```
module {
  tt.func public @add_kernel_0d1d2d3d(%arg0: !tt.ptr<f32> {tt.divisibility = 16 : i32},
    %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>) -> tensor<1024x!tt.ptr<f32>>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<
    %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tenso
    %13 = arith.addf %9, %12 : tensor<1024xf32>
    %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>

    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
    tt.return
  }
}
```

Triton IR中的优化pass

Canonicalizer Pass：规范化Pass

功能：将代码转换为一种规范化的形式，消除冗余和不必要的复杂结构，以便为后续的优化 Pass 提供便利。

执行命令：triton-opt canonicalizer_before.ttir -canonicalize &> canonicalizer_after.ttir

```
module {
  tt.func public @add_kernel_0d1d2d3d(%arg0: !tt.ptr<f32> {tt.divisibility = 16 : i32}, %a
    %c1024_i32 = arith.constant 1024 : i32
    %cst_0 = arith.constant 0 : i32
    %cst_1 = arith.constant 1 : i32
    %16 = arith.muli %c1024_i32, %cst_1 : i32
    %17 = arith.addi %c1024_i32, %cst_0 : 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>) -> tensor<1024x!tt.ptr<f32>>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<10
    %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<
    %13 = arith.addf %9, %12 : tensor<1024xf32>
    %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
    tt.return
  }
}
```

```
module {
  tt.func public @add_kernel_0d1d2d3d(%arg0: !tt.ptr<f32> {tt.divisibility = 16 : i32}, %arg1:
    %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>) -> tensor<1024x!tt.ptr<f32>>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf
    %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024
    %13 = arith.addf %9, %12 : tensor<1024xf32>
    %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
    tt.return
  }
}
```

Triton IR中的优化pass

执行命令：triton-opt inliner_before.ttir -inline &> inliner_after.ttir

Inliner Pass:

内联Pass

功能：直接将函数调用替换为函数体的代码，避免了函数调用所带来的开销，从而提高了程序的执行效率。

```
module {
  tt.func public @add_kernel_0d1d2d3(%arg0: !tt.ptr<f32> {tt.divisibil
    %0 = tt.get_program_id x : i32
    %c1024_i32 = arith.constant 1024 : i32
    %1 = arith.muli %0, %c1024_i32 : i32
    %2 = tt.make_range {end = 1024 : i32, start = 0 : i32} : tensor<10
    %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.make_range {end = 1024 : i32, start = 0 : i32} : tensor<10
    %8 = tt.splat %1 : (i32) -> tensor<1024xi32>
    %9 = arith.addi %8, %7 : tensor<1024xi32>
    %10 = tt.splat %arg3 : (i32) -> tensor<1024xi32>
    %11 = arith.cmpi slt, %9, %10 : tensor<1024xi32>
    %12 = tt.splat %arg0 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %13 = tt.addptr %12, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi
    %14 = tt.load %13, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile
    %15 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %16 = tt.addptr %15, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi
    %17 = tt.load %16, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile
    %18 = tt.call @add_operator__fp32S1024S_fp32S1024S__(%14, %17) : (
    %19 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %20 = tt.addptr %19, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi
    tt.store %20, %18, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<
    tt.return
  }
  tt.func private @add_operator__fp32S1024S_fp32S1024S__(%arg0: tensor
    %0 = arith.addf %arg0, %arg1 : tensor<1024xf32>
    tt.return %0 : tensor<1024xf32>
  }
```

```
module {
  tt.func public @add_kernel_0d1d2d3(%arg0: !tt.ptr<f32> {tt.divisibility =
    %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<1024xi3
    %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>) -> tensor<1024x!tt.ptr<f32>>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = fal
    %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = f
    %13 = arith.addf %9, %12 : tensor<1024xf32>
    %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024x
    tt.return
  }
```

Triton IR中的优化pass

Rewrite-tensor-pointer Pass：重写张量指针Pass

功能：识别并重写那些涉及张量指针（tensor pointers）的操作，以提高性能和内存使用效率。

执行命令：triton-opt rewrite_before.ttir -triton-rewrite-tensor-pointer &> rewrite_after.ttir

```
  %20 = arith.extsi %arg6 : i32 to i64
- // CHECK-NOT: tt.make_tensor_ptr
- %21 = tt.make_tensor_ptr %arg0, [%18, %19], [%20, %c1_i64], [%16, %17]
- %22 = arith.muli %15, %c32_i32 : i32
- %23 = arith.extsi %arg4 : i32 to i64
- %24 = arith.extsi %arg7 : i32 to i64
- // CHECK-NOT: tt.make_tensor_ptr
- %25 = tt.make_tensor_ptr %arg1, [%19, %23], [%24, %c1_i64], [%17, %22]
- %26 = arith.addi %arg5, %c31_i32 : i32
- %27 = arith.divsi %26, %c32_i32 : i32
- %28 = arith.index_cast %27 : i32 to index
- %29:3 = scf.for %arg9 = %c0 to %28 step %c1 iter_args(%arg10 = %cst, %
-   // CHECK: tt.load %{{.*}}, %{{.*}}, %{{.*}} {cache = 1 : i32, evict
```

```
  %20 = arith.extsi %arg6 : i32 to i64
+ %21 = arith.extsi %16 : i32 to i64
+ %22 = arith.extsi %17 : i32 to i64
+ %23 = arith.muli %15, %c32_i32 : i32
+ %24 = arith.extsi %arg4 : i32 to i64
+ %25 = arith.extsi %arg7 : i32 to i64
+ %26 = arith.extsi %17 : i32 to i64
+ %27 = arith.extsi %23 : i32 to i64
+ %28 = arith.addi %arg5, %c31_i32 : i32
+ %29 = arith.divsi %28, %c32_i32 : i32
+ %30 = arith.index_cast %29 : i32 to index
+ %31:5 = scf.for %arg9 = %c0 to %30 step %c1 iter_args(%arg10 = %cst, %ar
+   %57 = tt.splat %arg0 : (!tt.ptr<f16>) -> tensor<128x32x!tt.ptr<f16>>
+   %58 = tt.splat %arg11 : (i64) -> tensor<128xi64>
+   %59 = tt.make_range {end = 128 : i32, start = 0 : i32} : tensor<128xi3
+   %60 = arith.extsi %59 : tensor<128xi32> to tensor<128xi64>
+   %61 = arith.addi %58, %60 : tensor<128xi64>
+   %62 = tt.expand_dims %61 {axis = 1 : i32} : (tensor<128xi64>) -> tenso
+   %63 = tt.splat %20 : (i64) -> tensor<128x1xi64>
+   %64 = arith.muli %62, %63 : tensor<128x1xi64>
+   %65 = tt.broadcast %64 : (tensor<128x1xi64>) -> tensor<128x32xi64>
+   %66 = tt.addptr %57, %65 : tensor<128x32x!tt.ptr<f16>>, tensor<128x32x
+   %67 = tt.splat %arg12 : (i64) -> tensor<32xi64>
+   %68 = tt.make_range {end = 32 : i32, start = 0 : i32} : tensor<32xi32>
```

AdvancedCompiler
Tel: 13839830713