Triton编译器优化与FlagTree设计理念

北京智源人工智能研究院 刘笑妍

2025 年 8 月

01 Triton编译器优化与发散

优化现状、挑战与生态发散

02 Flagtree愿景与设计理念

多元AI芯片的算子编译器生态开源共建愿景

Flagtree设计理念

03 Flagtree架构及重要模块实现

架构overview

前端扩展、硬件抽象、dialect

Pass编写示例

01 Triton编译器优化与发散

优化现状、挑战与生态发散

02 Flagtree愿景与设计理念

多元AI芯片的算子编译器生态开源共建愿景

Flagtree设计理念

03 Flagtree架构及重要模块实现

架构overview

前端扩展、硬件抽象、dialect

Pass编写示例

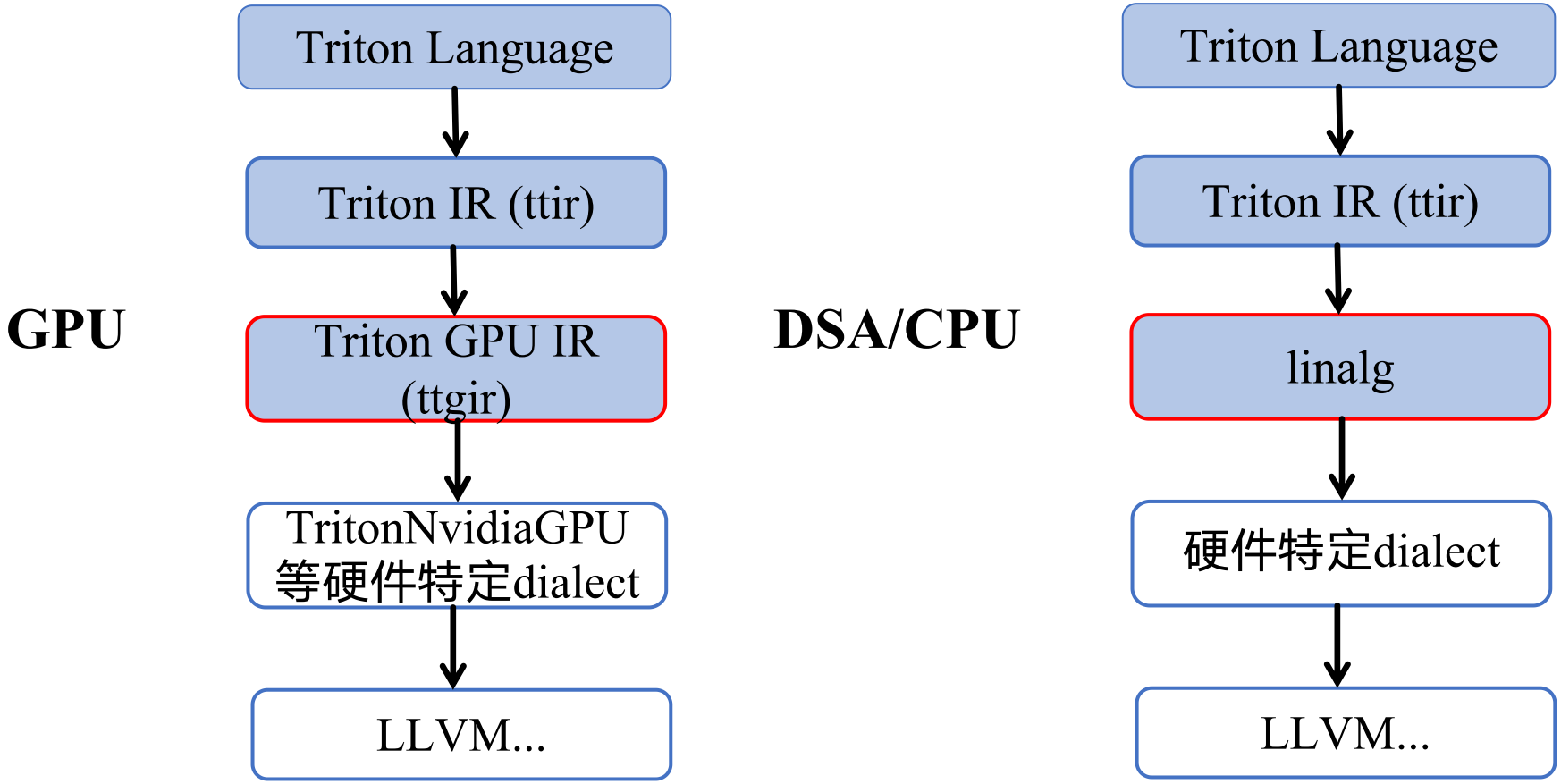
Triton编译器优化与发散

随着Triton生态的流行，多方向发展带来机遇的同时也面临诸多挑战，编译、语言、社区等多维问题逐步凸显

- 编译优化路径的发散：为了适配不同的后端架构，GPU、DSA与CPU间MLIR路径显著差异化，且同类型后端厂商间的实现也有发散趋势
- Triton 语言的发散：为了弥补Triton语言带来的性能损失（例如缺少硬件相关优化的显式控制、缺乏硬件特定调优参数需求）在Triton语言进行扩展
- 社区发展带来的生态蔓延：难以合并与跟进Triton主社区，同时语言层发散将逐步扩展到算子生态，致Triton生态分散，兼容与协作难度显著增加。

Triton编译器优化与发散

- **编译优化路径的发散：**为了**适配不同的后端架构**，GPU、DSA与CPU间MLIR路径显著差异化，且同类型后端厂商间的实现也有发散趋势



Triton编译器优化与发散

- **编译优化路径的发散：**为了适配不同的后端架构，GPU、DSA与CPU间MLIR路径显著差异化，且同类型后端**厂商间**的实现也有发散趋势

GPU

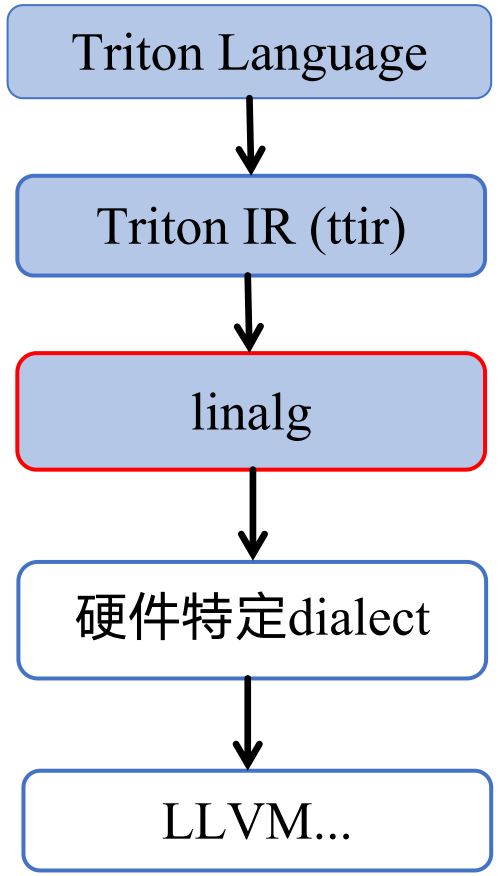
操作和优化		发散点
	Triton Language	
AST处理，少量循环相关等优化	Triton IR (ttir)	AST传入厂商特定参数 (intel引入warp层次dialect)
GPU相关并行分块，shared memory，异步拷贝	Triton GPU IR (ttgir)	各国产GPU厂商对此部分对op进行少量扩展
映射到具体硬件模块，例如TMA	TritonNvidiaGPU 等硬件特定dialect	
	LLVM...	

**GPU发散主要还是围绕着NVDIA GPU，可以由工程实现，后续同事会介绍*

Triton编译器优化与发散

- **编译优化路径的发散：**为了适配不同的后端架构，GPU、DSA与CPU间MLIR路径显著差异化，且同类型后端**厂商间**的实现也有发散趋势

DSA/CPU



Trion-linalg
寒武纪

处理/引入分块
ptr dialect

linalg extern

在linalg层进行了部分硬件相关优化，但扩展方法与硬件绑定

Trion-shared
e.g. aipu

处理/引入分块
ptr dialect

Trion-ascend
昇腾

不进行线程并行分块

linalg extern

硬件相关优化较少，性能优化主要在硬件dialect进行

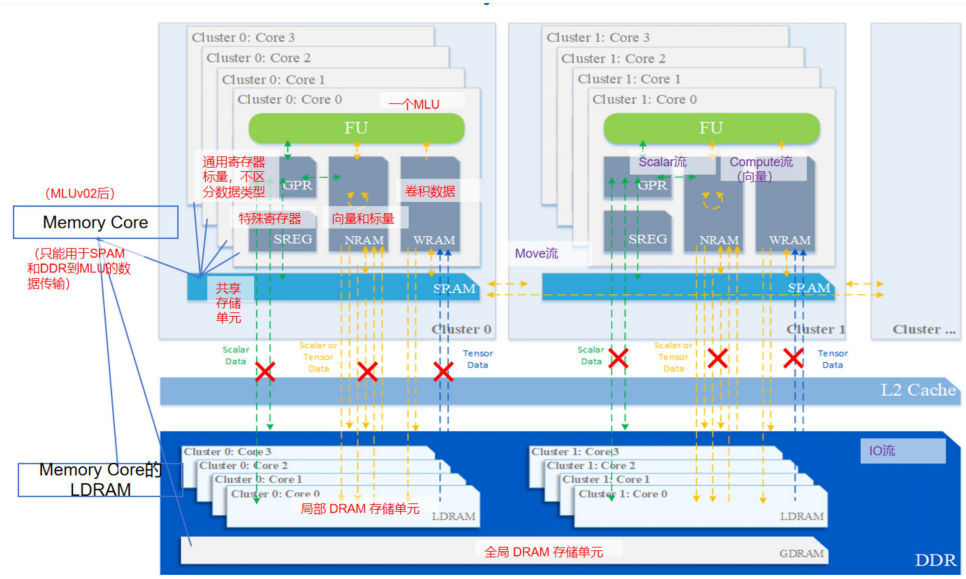
Triton编译器优化与发散

- **编译优化路径的发散：**为了适配不同的后端架构，GPU、DSA与CPU间MLIR路径显著差异化，且同类型后端**厂商间**的实现也有发散趋势

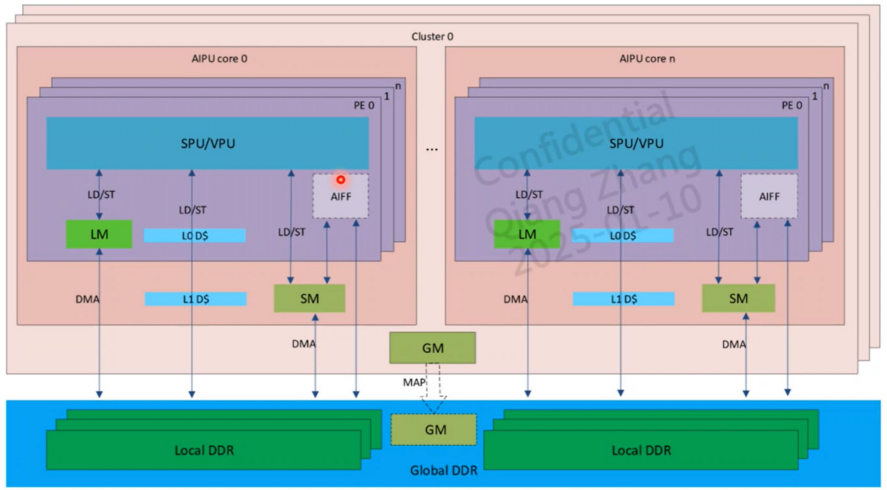
DSA尚未存在统一的dialect，且差异性大 (厂商/型号)

PE是否同构，PE层次，memory层次和通路，memory可见域，同步管理，launch方法....

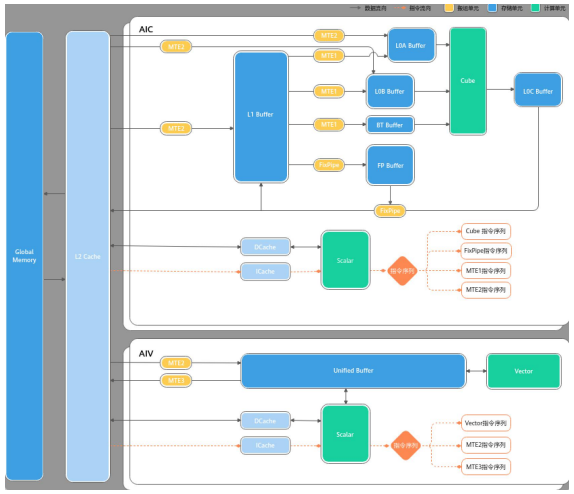
Trion-linalg
寒武纪



Trion-shared
e.g. aipu



Trion-ascend
昇腾



**DSA目前生态较为割裂，且有挑战性，为FLagTree关注的重点*

Triton编译器优化与发散

- **Triton 语言的发散：**为了弥补Triton语言带来的**性能损失**（例如缺少硬件相关优化的显式控制、缺乏硬件特定调优参数需求）在**Triton语言进行扩展**

Trion社区（Gluon）

```
if STAGE & 1:
    m_i, l_i0, l_i1, corr_bar, s_consumer, corr_producer, exp_turnstile = _softmax_inner_loop(  #
        tile_id, config, prog, s_consumer, corr_producer, exp_turnstile, corr_bar,  #
        offs_m, m_i, l_i0, l_i1, STAGE=4 - STAGE)
if STAGE & 2:
    m_i, l_i0, l_i1, corr_bar, s_consumer, corr_producer, exp_turnstile = _softmax_inner_loop(  #
        tile_id, config, prog, s_consumer, corr_producer, exp_turnstile, corr_bar,  #
        offs_m, m_i, l_i0, l_i1, STAGE=2)

if config.use_fadd2_reduce:
    l_i = l_i0 + l_i1
else:
    l_i = l_i0

s_tmem, s_bar, s_consumer = s_consumer.acquire()
m_i_tmem, l_i_tmem = _borrow_s_for_epilogue(config, s_tmem)
m_i_tmem.store(gl.convert_layout(m_i.expand_dims(1), config.alpha_2d_layout))
l_i_tmem.store(gl.convert_layout(l_i.expand_dims(1), config.alpha_2d_layout))
```

显式控制memory layout

厂商扩展（以triton-ascend为例）

```
for xoffset_sub in range(0, XBLOCK, XBLOCK_SUB):
    xindex = xoffset + xoffset_sub + tl.arange(0, XBLOCK_SUB)[:]
    xmask = xindex < xnumel
    x0 = xindex
    tmp0 = tl.load(in_ptr0 + (x0), xmask)
    tl.compile_hint(tmp0, "hint_a")
    tl.multibuffer(tmp0, 2)
    tmp2 = tmp0
    tl.compile_hint(tmp2, "hint_b", 42)
    tl.compile_hint(tmp2, "hint_c", True)
    tl.store(out_ptr0 + (xindex), tmp2, xmask)
```

在参数中加入hint

```
@pytest.mark.parametrize('param_list',
                         [
                             ['float32', (2, 4096, 8), 2, 32768, 1024],
                         ]
                         )
```

↓ 上述发散**必然带来**

- **社区发展带来的生态蔓延：**难以合并与跟进**Triton主社区**，同时语言层发散将逐步**扩展到算子生态**，致Triton生态分散，兼容与协作难度显著增加

01 Triton编译器优化与发散

优化现状、挑战与生态发散

02 Flagtree愿景与设计理念

多元AI芯片的算子编译器生态开源共建愿景

Flagtree设计理念

03 Flagtree架构及重要模块实现

架构overview

前端扩展、硬件抽象、dialect

Pass编写示例

Flagtree愿景与设计理念

多元 AI 芯片的算子编译器统一生态开源共建：生态、易用、性能

- **汇聚生态：**防止生态发散从编译器向算子层蔓延，以此为出发点
- **后端接入：**持续新增后端接入，类型涵盖各类芯片
- **仓库-中间层-编程接口统一：**以多元 AI 芯片统一编译器仓库为基础，对多后端从编程接口扩展到中间层转换进行横向对齐设计
- **生态-性能-灵活性：**既要保持和发展 Triton 生态，又要支撑特定硬件属性关联的编译指导和优化，并提升后端友好的算子编写的灵活性
- **最终目的是在生态、易用、性能上对各 AI 芯片提供算子库及编译器的软件栈支撑**

https://github.com/FlagTree/flagtree
https://gitee.com/flagtree/flagtree

Flagtree愿景与设计理念

前端：保守扩展，尽量避免引入新op和编程复杂性，保持原有triton编译器兼容

- 允许程序员通过注释嵌入硬件优化提示（flagtree_hints），对程序员使用成本低、生态兼容性好
- 实现性能提升，提升编译器可移植性、多平台统一的能力
- 加速编译器适配新需求速度、减少编译器负担

思路

- 前端：扩展Triton抽象语法树（AST）解析，将flagtree_hints编码为多级中间表示（MLIR）属性
- 中端：基于flagtree_hints属性设计优化过程，以增强优化效果
- 后端：使硬件供应商能够基于flagtree_hints选择性地注册过程
- 硬件抽象（unified hardware abstraction）：实现硬件特异信息记录，进行合法性检查

Flagtree愿景与设计理念

中端：基于linalg，尽量使用标注dialect，随演进按需提炼并扩展新op

- 存储层次、硬件单元不同，使得优化策略不同 → 机制策略分离设计
- 硬件约束和参数设置不同，使得下降pass需要大量的硬件特定参数（例如memory space编号）→ 引入硬件抽象（unified hardware abstraction）
- 在linalg层已经存在硬件特定的扩展（寒武纪triton-linalg）→防止生态进一步发散且保持兼容性

思路

- 按照ttir传入的hints作为策略进行性能优化
- pass根据unified hardware abstraction中每个硬件声明的参数进行转换或下降
- 尽量保持原有通用MLIR dialect（memref、affine、DMA等），同时设计FLIR扩展补足通用dialect不满足硬件厂商希望在上层做的操作

**后端：接入方式与Triton社区保持一致，保持third_party/和compile.py形式，后续同事会介绍*

01 Triton编译器优化与发散

优化现状、挑战与生态发散

02 Flagtree愿景与设计理念

多元AI芯片的算子编译器生态开源共建愿景

Flagtree设计理念

03 Flagtree架构及重要模块实现

架构overview

前端扩展、硬件抽象、dialect

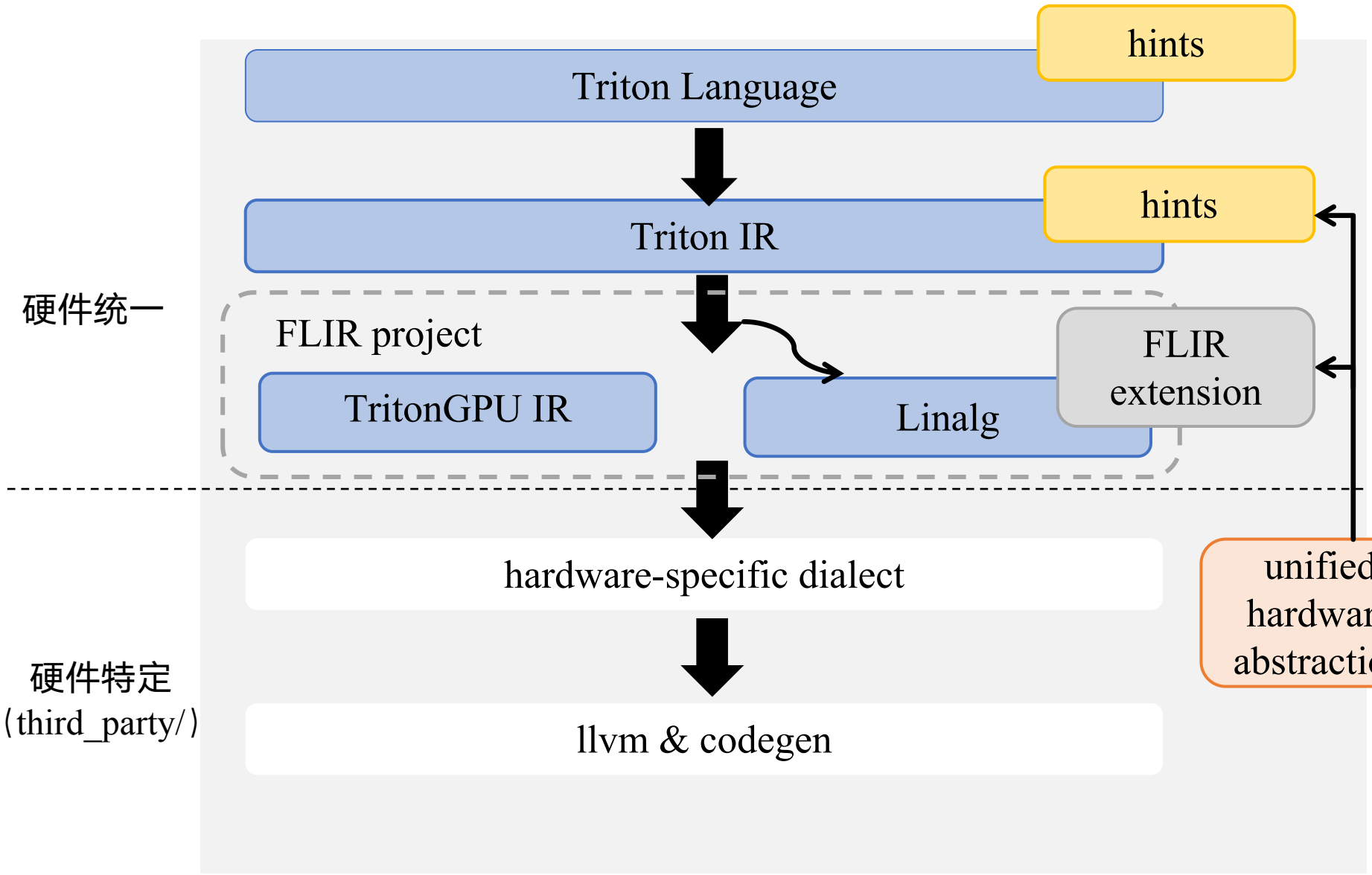
Pass编写示例

设计与演进中

持续厂商沟通

定期PMC会议

Flagtree架构及重要模块实现



- Flagtree hints

利用hints，使得triton language可以提供更多的性能指导

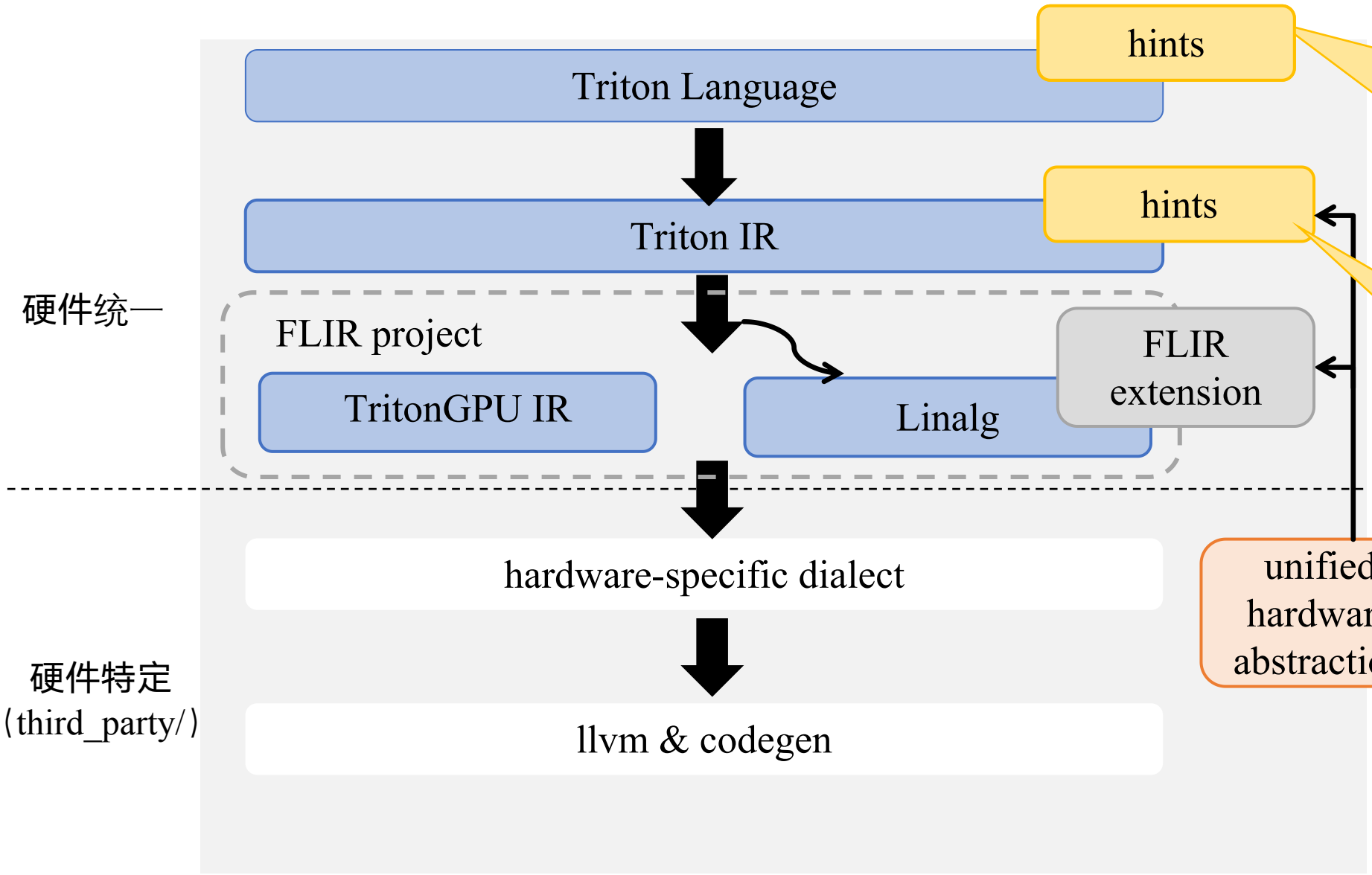
- FLIR扩展设计和下降

根据hints指导对ir进行 conversion 和 lowering

- 基于硬件抽象

类似配置文件，存储架构信息，不是IR

Flagtree架构及重要模块实现



hints

hints

FLIR extension

unified hardware abstraction

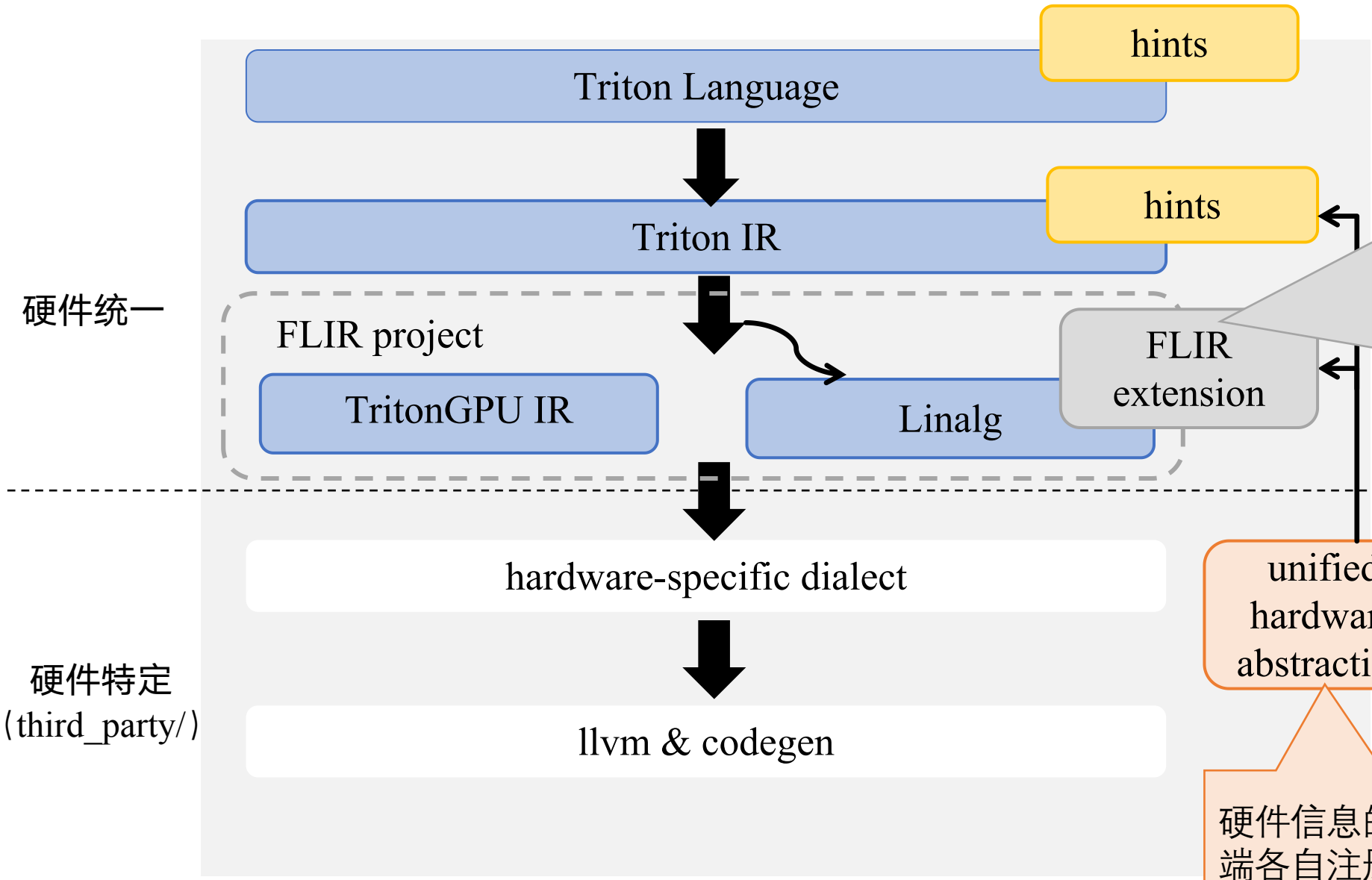
AST 解析扩展

- hints识别：解析#@hints注释，建立语法节点与对应提示之间的映射关系

TTIR Attribute 扩展

- 扩展TTIR Dialect，将hints内容作为attribute带入TTIR，后续pass将进行对应的优化或继续传递hints attribute

Flagtree架构及重要模块实现



Linalg Pass实现

承接hints（存在ttir的attribute中），基于linalg和unified hardware abstraction，实现DSA下降和共用优化的统一，并且有统一GPU的潜力

目前规划的优化主要包括：
1. 并行分块
2. 存储管理
3. DMA异步

前端语言扩展

- 语法: #@hints: 后面紧跟着注释指导内容，需要为特定的字符串

- 注释分为两类：
  - 硬件单元映射有关：指导数据存储、并行分配等，例如共享内存的分配
  - 编译优化有关：帮助编译器选择合适的优化策略、优化参数等，例如pipeline阶段数

- Example:

```
x = tl.load(x_ptr + offsets, mask=mask)  # @hint: shared_memory
y = tl.load(y_ptr + offsets, mask=mask)  # @hint: shared_memory
```

Flagtree架构及重要模块实现

AST 解析扩展

- hints识别：解析#@hints注释，建立语法节点与对应提示之间的映射关系
- hints前端验证：
  - 初步进行合法性检查，例如验证hints与目标架构是否匹配
  - 静默忽略无效提示，以确保编译成功

TTIR Attribute 扩展

- 扩展TTIR Dialect，将hints内容作为attribute带入TTIR，后续pass将进行对应的优化或继续传递hints attribute

```
def TT_LoadOp : TT_Op<"load", [
  ...
]> {
    let summary = "Load from a tensor of pointers or from a tensor pointer";

    let arguments = (
      ins
      AnyTypeOf<[TT_PtrLike, TT_TensorPtr]>:$ptr,
      Optional<TT_BoolLike>:$mask,
      Optional<TT_Type>:$other,

      DefaultValuedAttr<DenseI32ArrayAttr, "::llvm::ArrayRef<int32_t>{}">:$boundaryCheck,
      OptionalAttr<TT_PaddingOptionAttr>:$padding,
      DefaultValuedAttr<TT_CacheModifierAttr, "::mlir::triton::CacheModifier::NONE">:$cache,
      DefaultValuedAttr<TT_EvictionPolicyAttr, "::mlir::triton::EvictionPolicy::NORMAL">:$evict,
      DefaultValuedAttr<BoolAttr, "false">:$isVolatile,
      // TODO: now flagtree_hints is string, default value of an empty string (""), needed redesign
      DefaultValuedAttr<StrAttr, "\"\"">:$flagtree_hints
    );
  ⌘L to chat, ⌘K to generate
```

Flagtree架构及重要模块实现

硬件抽象 Unified Hardware Abstraction

- 在Flagtree/include实现基类，各后端在third_party/进行重写，使用宏定义拼接编译到项目

flagtree > include > flagtree > Common > C UnifiedHardware.h

```
14
15  class UnifiedHardware {
16
17  public:
18    ~UnifiedHardware() = default;
19    UnifiedHardware() = default;
20  #ifdef FLAGTREE_BACKEND
21    static bool registered;
22    int getDMATag();
23    int getSharedMemoryTag();
24    std::string getFlagTreeBackend() { return FLAGTREE_BACKEND; }
25  #else
26    static constexpr bool registered = false;
27    void *getDMATag() { return nullptr; }
28    void *getSharedMemoryTag() { return nullptr; }
29    std::string getFlagTreeBackend() { return "default"; }
30  #endif
31  };
32
33  std::unique_ptr<UnifiedHardware> createUnifiedHardwareManager();
34
35
```

triton include定义统一接口

flagtree > third_party > aipu > lib > Registrar.cc

```
1  #include "flagtree/Common/UnifiedHardware.h"
2  FLAGTREE_REGISTRAR(DMATag, 11)
3    ⌘L to chat, ⌘K to generate
```

third party后端各自注册具体信息

Flagtree架构及重要模块实现

FLIR pass扩展

处理memory space（以DMA tag为例，pass直接使用getDMATag()方法）

lib/Conversion/MemrefCopyToDMA_FlagTree/MemrefCopyToDMAFlagTree.cpp

单独的Flagtree pass

```
+          if (constOp.value() == 1)
+            continue;
+
+        if (auto intOp = val.getDefiningOp<arith::ConstantIntOp>())
+          if (intOp.value() == 1)
+            continue;
+      }
+      return false;
+    }
+    return true;
+  }
+
+  LogicalResult rewriteCopyToDma(memref::CopyOp op, OpAdaptor adaptor,
+                                 ConversionPatternRewriter &rewriter) const {
+
+    auto hardwareManager = mlir::flagtree::createUnifiedHardwareManager();
+    auto dmaTag = hardwareManager -> getDMATag();
+    if (!dmaTag) return failure();
+
+    Location loc = op.getLoc();
+    Value src = adaptor.getSource();
+    Value dst = adaptor.getTarget();
+
+    Value zero = rewriter.create<arith::ConstantIndexOp>(loc, 0);
+
+    SmallVector<Value> srcIndices, dstIndices;
+    Value numElements;
+    Operation *srcDef = src.getDefiningOp();
+    //
```

从我们的manager得到信息

做非法保护，如果失败则不修改ir

Flagtree架构及重要模块实现

北京智源人工智能研究院
BEIJING ACADEMY OF ARTIFICIAL INTELLIGENCE

FLIR pass扩展

基于hints实现shared memory使用，开发中，代码将进行Unified Hardware Abstracttion重构（正在进行memory layout相关调研和厂商需求收集）

4 lib/AnalysisStructured/PtrAnalysis.cpp

```
@@ -1122,6 +1122,10 @@ LogicalResult PtrAnalysis::rewriteLoadOp(triton::LoadOp op,
1122 1122
1123 1123     auto loadOp = builder.create<tts::LoadOp>(loc, ptr, dims, scalarOther);
1124 1124
1125 +   if (op->getAttr("flagtree_hints")) {
1126 +     loadOp->setAttr("flagtree_hints", op->getAttr("flagtree_hints"));
1127 +   }
1128 +
```

找到有hints的load op

lib/Conversion/StructuredToMemref/StructuredToMemref.cpp

```
+   auto tensorType = cast<RankedTensorType>(op.getType());
+   auto shape = tensorType.getShape();
+   SmallVector<OpFoldResult> offsets = reinterpretOp.getMixedOffsets();
+   SmallVector<OpFoldResult> tensorShape, modShape;
+   for (int64_t dim : tensorType.getShape()) {
+     tensorShape.push_back(rewriter.getIndexAttr(dim));
+     int64_t sharedDim = ShapedType::isDynamic(dim) ? dim : dim * 4;
+     modShape.push_back(rewriter.getIndexAttr(sharedDim));
+   }
+   SmallVector<OpFoldResult> strides(tensorType.getRank(),
+                                     rewriter.getIndexAttr(1));
```

每个后端alloc shared mem大小的逻辑不同

```
} else {
  rewriter.create<memref::CopyOp>(loc, ptr, alloc);
  if (cast<MemRefType>(alloc.getType()).getMemorySpaceAsInt() == 8) {
    // tensorSubview represents moving data to the space of the
    // corresponding TEC in shared memory and converting it into tensor form
    SmallVector<OpFoldResult> strides(tensorType.getRank(),
                                      rewriter.getIndexAttr(1));
    auto tensorSubview = createTensorSubview(op, ptr, alloc, rewriter);
    // dstSubview represents moving to the
    // specified TEC space of shared memory
    auto modOffsets = tensorSubview.getMixedOffsets();
    auto allocType = cast<MemRefType>(alloc.getType());
    auto dstType = memref::SubViewOp::inferResultType(allocType, modOffsets,
                                                      mixedDims, strides);
    auto dstSubview = rewriter.create<memref::SubViewOp>(
        loc, cast<MemRefType>(dstType), alloc, modOffsets, mixedDims,
        strides);
    rewriter.create<memref::CopyOp>(loc, srcSubview, dstSubview);
    Value tensor = rewriter.create<bufferization::ToTensorOp>(
        loc, tensorType, tensorSubview, true /* restrict */,
        true /* writable */);
    rewriter.replaceOp(op, tensor);
  } else {
    rewriter.create<memref::CopyOp>(loc, srcSubview, dstSubview);
    if (cast<MemRefType>(alloc.getType()).getMemorySpaceAsInt() == 8) {
      // The tensorSubview passes bufferization.to_tensor to
      // convert memref into tensor form for subsequent computations
      SmallVector<OpFoldResult> strides(tensorType.getRank(),
                                        rewriter.getIndexAttr(1));
      auto tensorSubview = createTensorSubview(op, ptr, alloc, rewriter);
      // dstSubview represents moving to the
      // specified TEC space of shared memory
      auto modOffsets = tensorSubview.getMixedOffsets();
      auto allocType = cast<MemRefType>(alloc.getType());
      auto dstType = memref::SubViewOp::inferResultType(allocType, modOffsets,
                                                        mixedDims, strides);
      auto dstSubview = rewriter.create<memref::SubViewOp>(
          loc, cast<MemRefType>(dstType), alloc, modOffsets, mixedDims,
          strides);
      rewriter.create<memref::CopyOp>(loc, srcSubview, dstSubview);
      Value tensor = rewriter.create<bufferization::ToTensorOp>(
          loc, tensorType, tensorSubview, true /* restrict */,
          true /* writable */);
      rewriter.replaceOp(op, tensor);
    } else {
      rewriter.create<memref::CopyOp>(loc, srcSubview, dstSubview);
      rewriter.replaceOp(op, alloc);
    }
  }
```

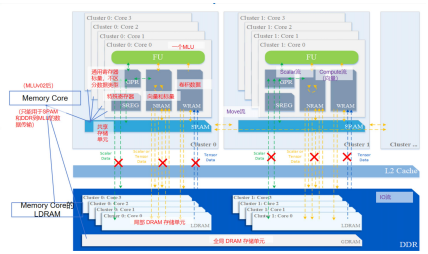
不同的memory space编号

根据不同后端的shared mem alloc逻辑，需要实现不同的ptr重写

Flagtree架构及重要模块实现

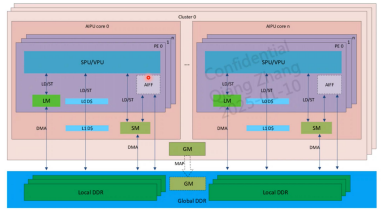
北京智源人工智能研究院
BEIJING ACADEMY OF ARTIFICIAL INTELLIGENCE

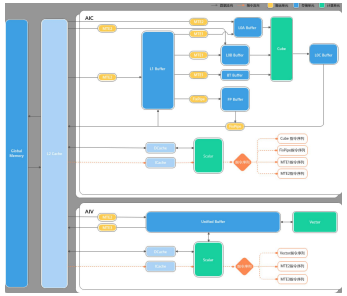
FLIR dialect扩展 （正在进行linalg层op 扩展调研和厂商需求收集）



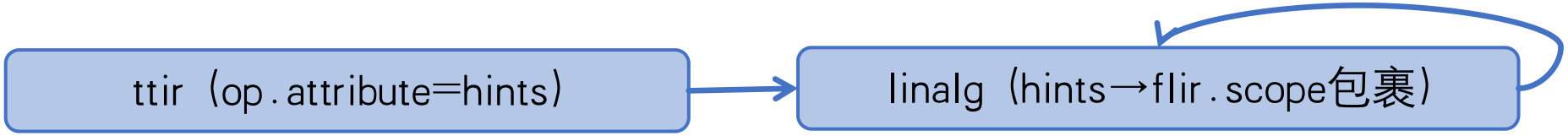
- 初步设计
  - flir.scope{“字符串标记信息”}{通用MLIR代码}

```
flir.scope{"tiling{sizePerThread=[2,2],threadsPerWarp=[16,2]...}..."}{
    memref.load...
    linalg.matmul...
    memref.store...
}
```





  - 实现OpTrait::HasIsolatedFromAbove，让已有pass能进入内部
  - 利用Unified Hardware Abstraction注册的规则进行下降，若没有则保持
- 先用flir.scope方式作为信息补足，同时加速开发进度。等到出现收敛迹象后统一注册为新的flir op



Unified Hardware Abstraction

Thanks!

个人邮箱：xyliu01@baai.ac.cn

欢迎加入FLagTree开源项目！

电话：010 - 6893 3383

地址：北京市海淀区成府路150号智源大厦

智源公众号