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

Triton算子编写实战

北京智源人工智能研究院

2025年08月17日

内容

01 实操说明（5min）

02 算子任务1（10 min）

03 算子任务2 （30 min）

04 算子集成 （10 min）

实操说明

- 硬件
  - A100 单卡（演示卡）
- 软件
  - Torch 2.6.0 & Triton 3.2.0
- 超算环境
  - 共享目录：
    - N32-E 路径： /home/bingxing2/apps/flagtree
    - N76H8 路径：/data/public/flagtree
  - singularity shell --nv -w flags
  - source /opt/venv/flagtree/bin/activate
- 个人环境
  - NVIDIA 的 ngc pytoch 镜像即可，包含所有需要软件

```
[gpuuser992@ln01 flagtree]$ ll
total 58798776
drwxr-xr-x 19 gpuuser001 gpuuser001        8192 Aug 14 20:57 123
drwxrwxr-x  3 gpuuser001 gpuuser001        4096 Aug 17 11:16 code
-rwxr-xr-x  1 gpuuser001 gpuuser001 12607320064 Aug 15 16:05 flags.sif
drwxrwxr-x 16 gpuuser001 gpuuser001        8192 Aug 14 20:51 flagtree
-rwxr-xr-x  1 gpuuser001 gpuuser001 11502166016 Aug 14 20:49 flagtree.sif
-rw-rw-r--  1 gpuuser001 gpuuser001 21448412160 Aug 14 20:40 flagtree.tar
-rw-rw-r--  1 gpuuser001 gpuuser001 14652027465 Aug 14 20:45 flagtree.tar.gz
-rw-rw-r--  1 gpuuser001 gpuuser001           5 Aug 15 16:43 test
[gpuuser992@ln01 flagtree]$ pwd
/home/bingxing2/apps/flagtree
```

如何认定线下上机成功?

- Step1: 跟着讲师实操上机开发算子
- Step2: 将最终源码&源码跑通的截屏，发送至邮箱：liumin@baai.ac.cn
  - 邮件命名要求：姓名+单位+上机认证
- Step3: 智源研究院讲师确认后，认定上机成功

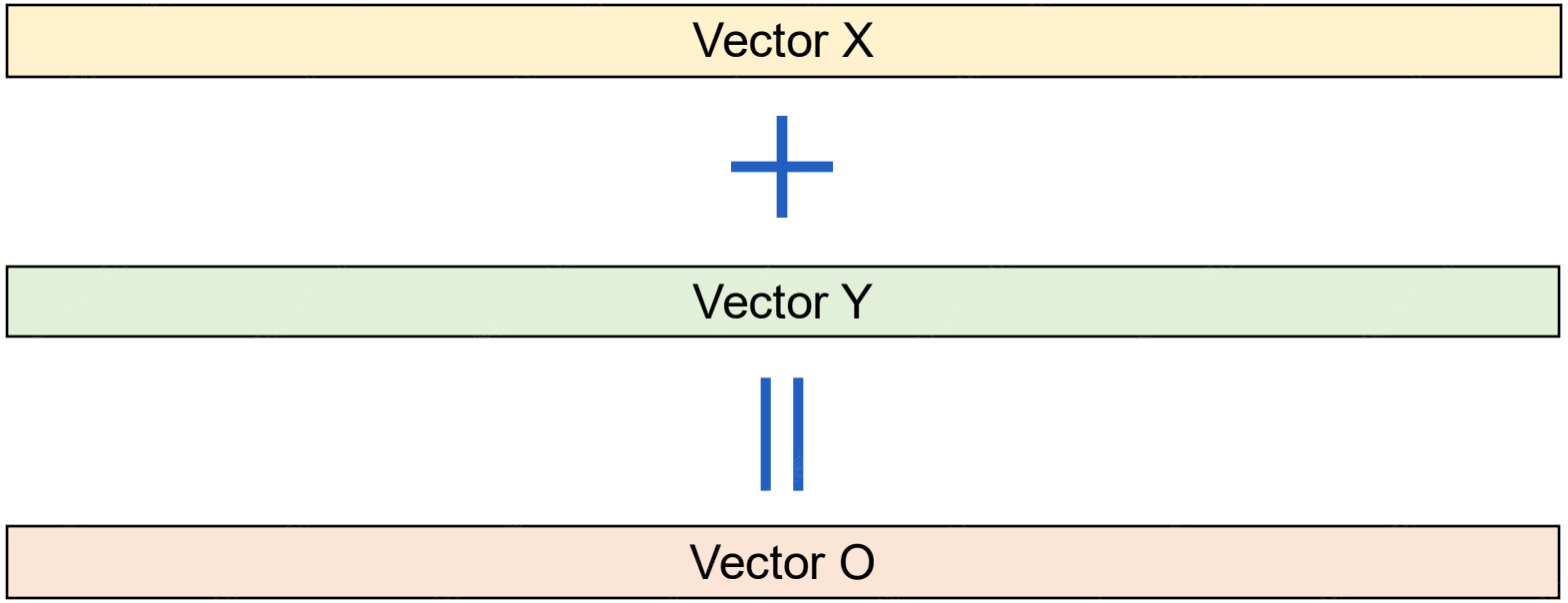
概念复习

- CTA（cooperation thread array）
  - aka thread block，是 Triton 程序的基本执行单元。
  - Triton 编程是 CTA 级别的编程。一个 program 表示一个 CTA 的计算逻辑。
  - CTA 以下的细节和优化都由编译器处理。开发者主要关心 grid 的任务划分，并发 Kernel 调度等。

- Tile
  - 数据划分的基本抽象。
  - 一个 CTA 可以处理一个 tile，也可以循环处理多个 tile。
  - Tile 大小的选择直接影响并行度和性能。

- wrapper vs kernel
  - kernel：描述所有 CTA 的计算逻辑，通过 program_id 区分不同 CTA。
  - wrapper：在 Python 侧决定 grid 大小、分块方式，并调用 Kernel。

算子任务1 - VectorAdd

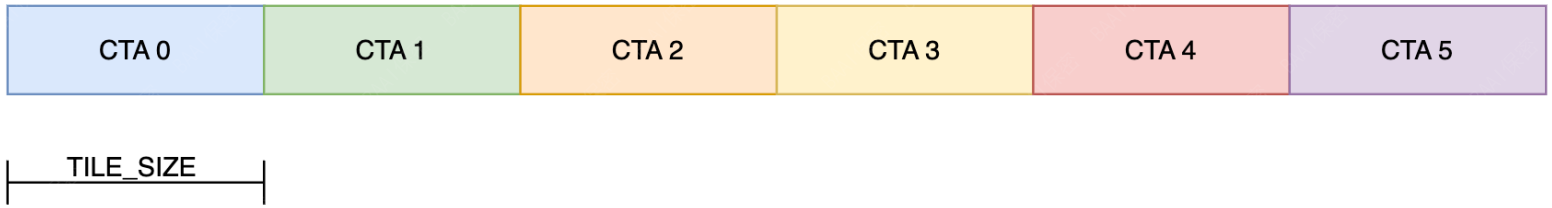


简易场景：

- 两个向量对位相加
- 向量数据类型相同、所在设备相同、形状相同
- 向量均连续存储，尺寸适中

算子任务1 - VectorAdd

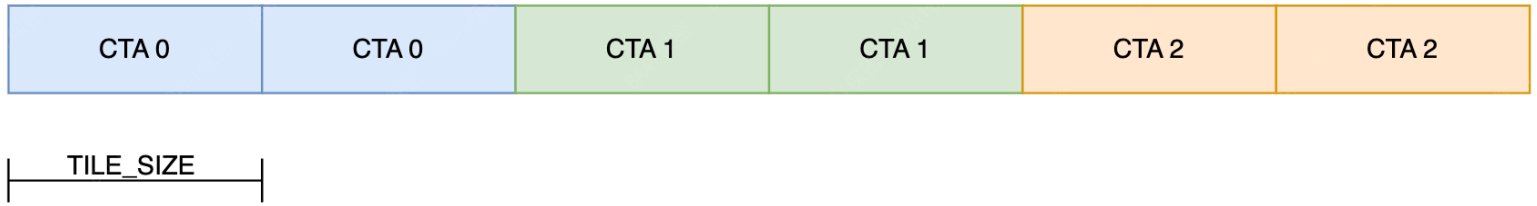
- 任务划分方案1
  - 每个CTA 算1个TILE



- 基本步骤
  - 数据索引计算 & 掩码计算
  - 数据加载
  - 核心计算逻辑
  - 结果存储

算子任务1 - VectorAdd

- 任务划分方案2
  - 每个CTA 算2个TILE

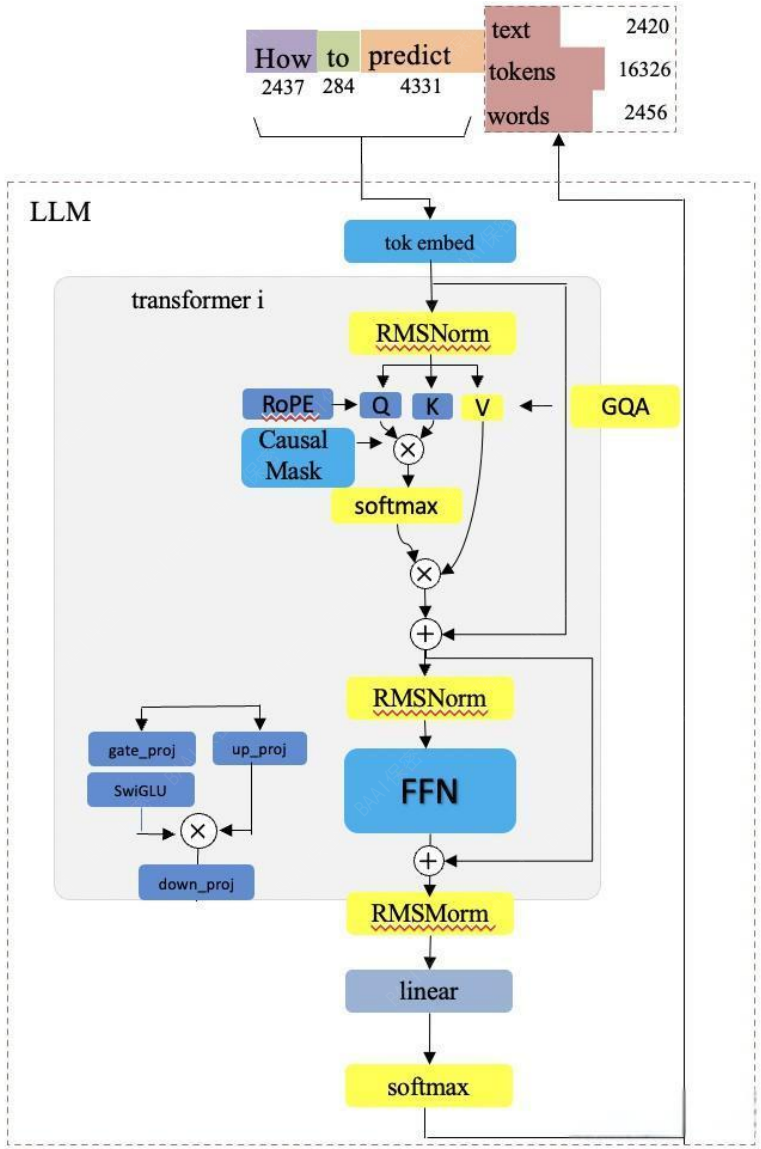


- 基本步骤
  - for i in range(tilers_per_cta):
    - 数据索引计算 & 掩码计算
    - 数据加载
    - 核心计算逻辑
    - 结果存储

算子任务2 - RmsNorm

- RmsNorm
  - RMSNorm（Root Mean Square Layer Normalization）是一种常用的归一化方法，在 Transformer 模型中被广泛应用。
  - 在实际使用中，RMSNorm 的输入通常是形状为 [batch, seq_len, hidden_size] 的张量，归一化操作发生在最后一维（即 hidden_size 维度）。
  - 其数学公式如下：

$$y_i = \frac{x_i}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i)^2 + \epsilon}} * \gamma$$



算子任务2 - RmsNorm

- 数学公式

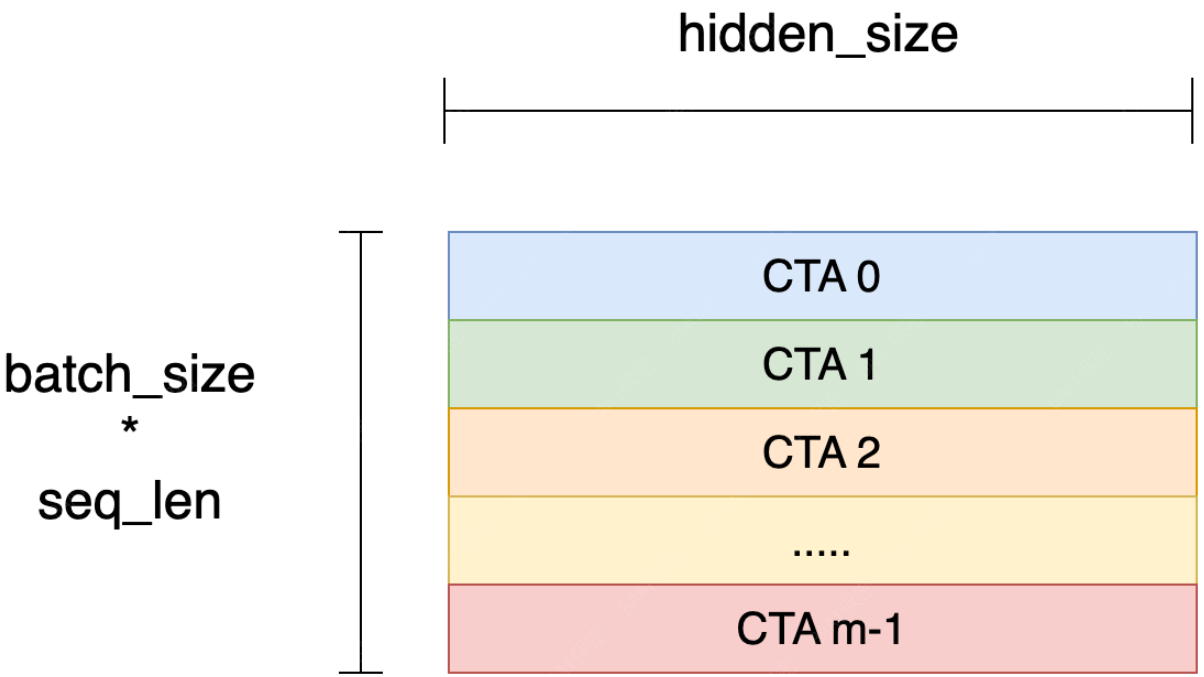
$$y_i = \frac{x_i}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i)^2 + \epsilon}} * \gamma$$

- 计算类型分析
  - 首先，需要沿着某一维（通常是最后一维 hidden_size) 计算平方和，再求均值并开方，得到RMS值。这一步属于归约（reduction）操作。
  - 接着，利用这个 RMS 值对每个元素进行归一化（元素级除法），再乘以缩放权重（元素级乘法）。这两步都属于元素级（elementwise）操作。
  - 整体而言，RMSNorm 的计算过程是 reduction + elementwise 的组合。

算子任务2 - RmsNorm

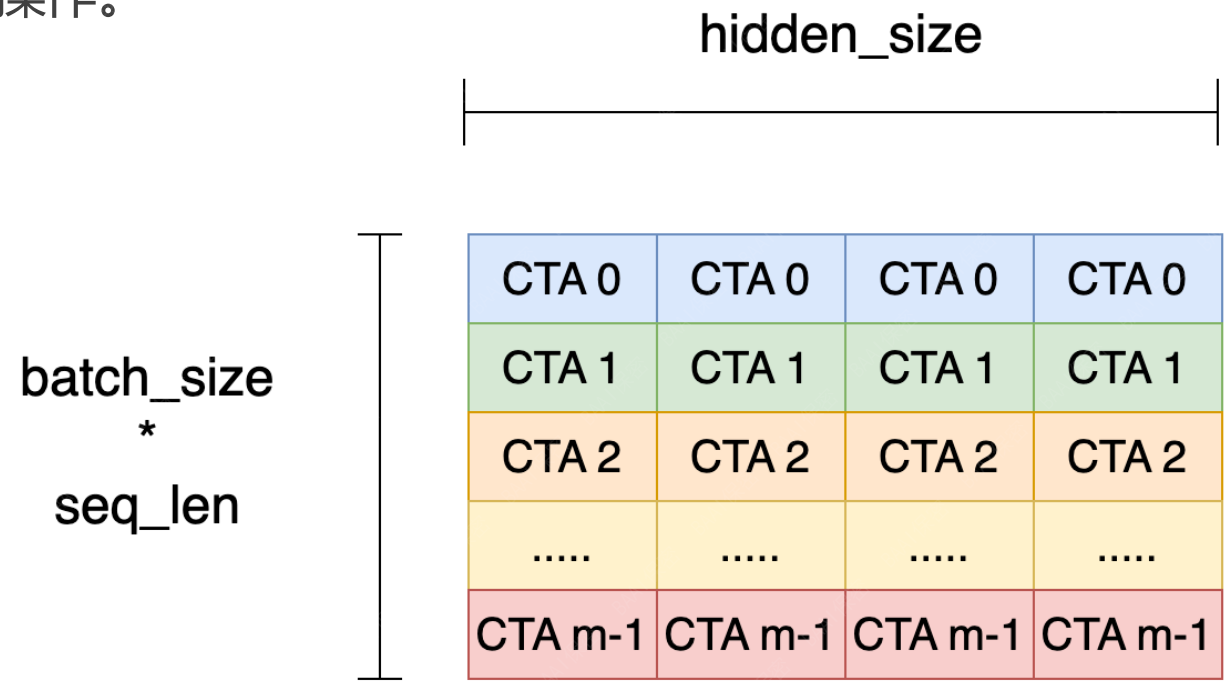
- 任务划分方案1
  - 由于需要在最后一维（hidden_size）上进行 reduction，因此可以将任务划分为 **每个 CTA 负责处理一行向量。**
  - 整个计算任务就被划分为 batch_size × seq_len 个并行单元（pid），每个单元独立完成对应行的归约与后续元素级计算。
  - 设置 BLOCK_SIZE > hidden_size

- 简易场景
  - 向量数据类型相同、所在设备相同
  - 形状符合常规定义
  - 向量均连续存储，尺寸适中
  - 只考虑推理场景（前向）



算子任务2 - RmsNorm

- 任务划分方案2
  - 当 hidden_size 很大时，如果让每个 CTA 处理整个行向量（即 TILE_SIZE == hidden_size），每个 SM 内的线程需要一次性读取大量数据，这可能导致寄存器压力过高甚至影响性能。
  - 在 **CTA 内进一步划分数据块**，让 CTA 通过循环多次处理每个 block_size 的子块，从而完成整个 hidden_size 的归约与归一化操作。



算子任务2 - RmsNorm

- 任务划分方案 3：
  - 场景：hidden_size 很大，batch_size × seq_len 很小
  - 问题：不同 SM 之间计算任务分配不均，部分 SM 负载过重，而其他 SM 空闲。
  - 解决方案：让**多个 CTA 协同处理同一行向量**，每个 CTA 负责 hidden_size 的一个子块，然后在 CTA 之间做归约汇总（通过全局原子操作实现）。

- 任务划分方案 4：
  - 场景：hidden_size 很小，batch_size × seq_len 很大
  - 问题：单行向量很短，单个 CTA 处理一行难以充分利用 SM 并行度。
  - 解决方案：让**一个 CTA 同时处理多行**（multi-row per CTA），在 CTA 内部通过循环或并行方式计算多行的 RMS 并归一化。

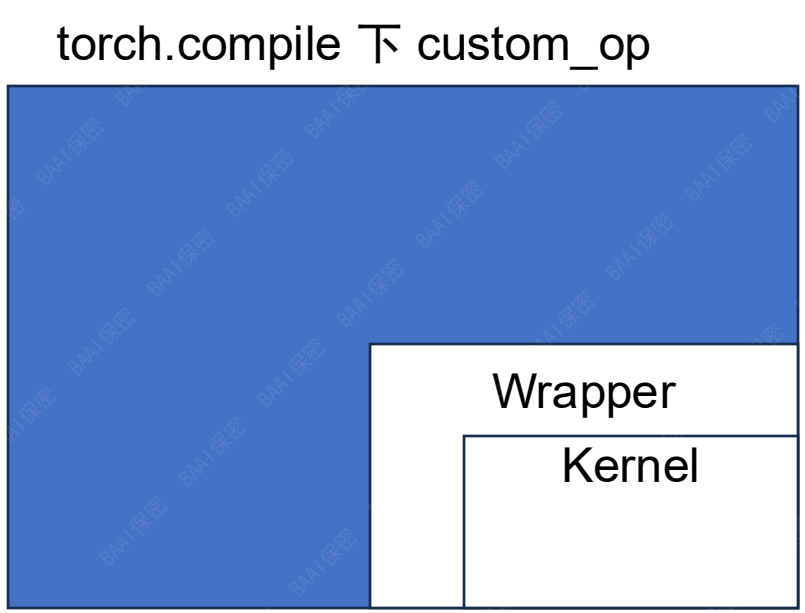
算子集成

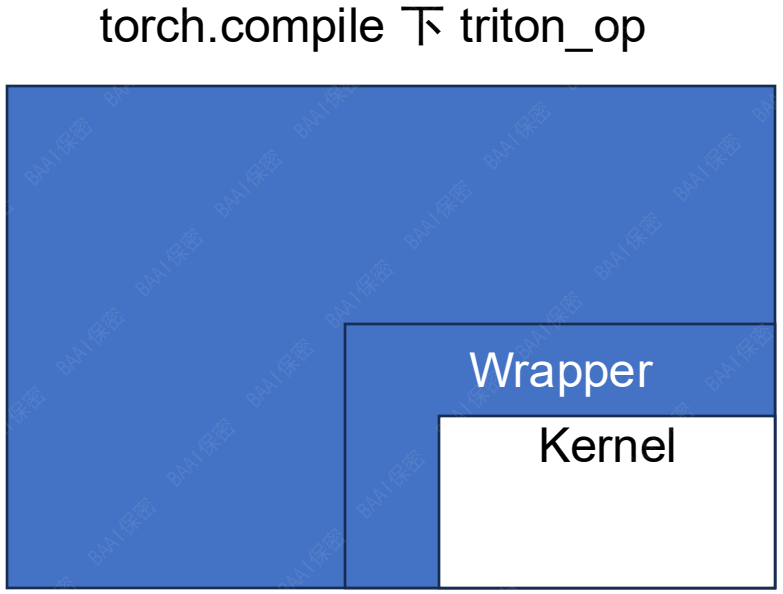
- 方式一： torch.library.Library + torch.library.define + torch.library.impl
  - 经典的算子注册方式，vLLM 框架使用该方式集成自定义算子。
  - torch.library.Library 提供了一个 handle，用于在已有库中声明和注册算子。它支持三种库类型（DEF / IMPL / FRAGMENT）
  - torch.library.define 声明算子的 schema
  - torch.library.impl 绑定算子的实现（Python 或 C++）

- 方式二： torch.library.custom_op 系列注册方式 （样例代码里有相应实现）
  - 当前官方推荐的算子注册方式 （Pytorch >= 2.4），只有python 版本API
  - Highlight：天然兼容 torch.compile，不会破坏图捕获流程
  - 适合在 Python 端快速注册自定义算子

算子集成

- 方式三：torch.library.triton_op 系列注册方式
  - 最新推出的算子注册方式，专门为 Triton kernel 设计。
  - 设计目标是缓解 Python wrapper 带来的性能开销。
  - Highlight：与 torch.compile 兼容，能够让torch.compile 优化直接作用到 wrapper。
  - Concerns：目前方案仍不成熟，容易踩坑，需要谨慎使用。





感谢垂听！

欢迎访问开源社区：

https://github.com/FlagOpen/FlagGems

Triton中国社区微信群