# 工大标志

# GPU计算实验报告

## 实验二 基于昇腾CANN的TBE算子开发（TIK）

学院：计算机科学与技术学院

姓名：钱泽凯

学号：1190202011

### 一、实验预习（10分）

1. 注册华为云账号：https://www.huaweicloud.com/
2. 课程内容预习：<https://www.hiascend.com/edu/courses>



图1 昇腾学院异构计算架构CANN中TBE算子开发（高级）



图2 本次实验主要涉及模块一到模块五

1. 问题（10分）：

（回答请使用红色文字）

1. 昇腾芯片（310或910）中的达芬奇架构具体是哪一部分？（ B ）

A、整颗芯片 B、AI CPU C、AI Core D、DVPP

答：AI Core 是昇腾Al芯片的计算核心，负责执行矩阵、向量、标量计算密集的算子任务，采用达芬奇架构。

1. 昇腾310中有2个AI Core，8个ARM A55（有一部分部署为AI CPU）。算子开发也有两类，使用TBE算子开发运行在AI Core上，使用AI CPU算子开发运行在AI CPU上。什么样的任务适合在AI Core上执行，什么样的任务适合在AI CPU执行？

答：

AI Core适合执行：负责执行矩阵、向量、标量计算密集的算子任务

AI CPU适合执行：负责执行不适合跑在Al Core上的算子(承担非矩阵类复杂计算)

1. Unified Buffer(UB)的size为多少？最小访问粒度？

答：

size为：256KB

最小访问粒度为：32B

1. 向量计算单元一次能够处理的数据量为多少？（A）

A、512字节 B、256字节 C、128字节 D、64字节

答：每个cycle可以完成两个128长度fp16类型的向量相加/乘， 或者64个fp32/int32类型的向量相加/乘，即512字节

1. 简述TIK算子开发过程

答：

1. 算子分析

进行算子内容的分析，随后设计schedule流程。

1. 算子实现

首先进行Python模块导入，然后定义目标机,并构建TIK DSL容器，进行数据定义并将数据搬入AI Core，在其中进行数据计算，最后数据搬出AI Core。

1. 算子编译

编译为CCEC代码，对编程者不可见，最后CCE编译器将其转为2进制比特流。

## 二、实验目标

1. 了解Davinci架构及其组成
2. 了解核上的计算单元和存储单元以及它们之间的数据流向
3. 了解什么是TIK及它的特点
4. 分析一个简单的TIK程序
5. 了解TIK数据操作和处理
6. 使用TIK的API接口完成一个算子开发

## 三、实验内容

完成链接中的实验：

https://www.hiascend.com/zh/college/onlineExperiment/codeLabTbeTik/tiks



图3 TIK算子开发实验

参考文档：

https://support.huaweicloud.com/tbedevg-cann503alpha2training/atlastikapi\_07\_0001.html



图4 CANN官方文档，TBE TIK相关API

## 四、实验代码（40分）

1. 实验分析（10分）

（回答使用红色文字）

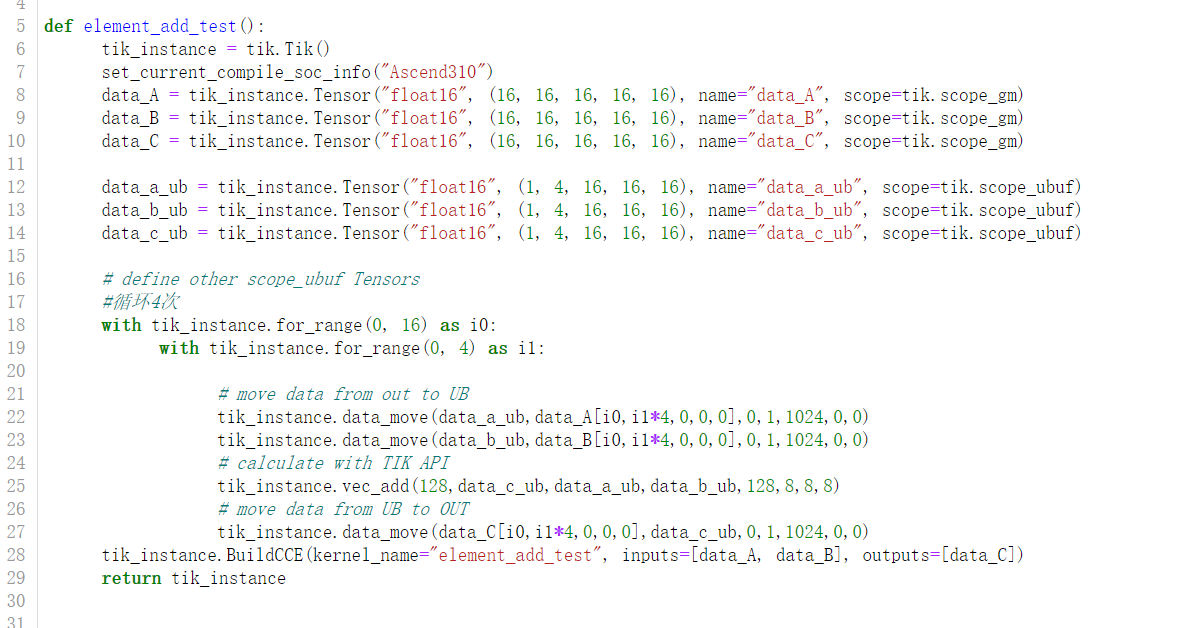
我们在代码中使用了for\_range，多次从内存向UB搬运数据，然后计算，为什么？

答：每个AICore总共需要计算50MB的数据，但UB可用大小一共才256KB，所以肯定是要走多次数据搬进UB，计算，数据搬出UB的过程。

2. 实验代码（30分）

（粘贴对应的完整代码截图）

四层循环方法：



（删掉此图，并粘贴完整代码截图）

## 五、实验结果（40分）

（运行结果截图）



## 六、TIK商用算子开发（10分）

阅读一个Vadd商用算子开发案例：

<https://www.hiascend.com/zh/college/onlineExperiment/detail/664612>

我们会看到在一个TIK商用算子中，即使是最简单的Vadd算子，也有很多要考虑的细节问题，列举一些细节并简单介绍。

（除下面示例以外，给出1个并简单介绍，合理即给3分；2个6分；3个及3个以上10分）。

示例：

1. for\_range参数使用注意

for\_range有多核（block\_num）和多线程（thread\_num）两个参数，不可同时使用，在外部大循环使用block\_num，内部小循环使用thread\_num

1. 开启多核，在多核循环内使用到的tensor，必须要定义在多核循环内

因为多核循环外和多核循环内的tensor内存分配同时从0开始分配，可能导致地址重叠，最后导致数据错误

1. 当数据量较大时，分阶段调用计算指令。

每个AICore总共需要计算50MB的数据，但UB可用大小一共才248KB，所以肯定是要走多次数据搬进UB，计算，数据搬出UB的过程。

我们依次需要计算在AI Core上一共要做多少次循环（一次循环指数据搬进AI Core，做计算，数据搬出AI Core的全过程）。每个循环的计算逻辑，计算上边的循环全结束后，还剩多少数据没有计算。对最后一点数据再做一次循环。

重点关注“burst”这个参数，这个参数的取值是0-65535，指的是一次搬运多少个“block”（也就是32字节）到目的地。65535这个数值足够大了，不需要搬运多次了，所以nburst直接取1即可。而“burst”这个参数则需要计算一下：用总传进来的总数据量除以每个block存放多少个元素，再向上取整就能得出需要多少个burst了

1. 分阶段调用计算指令的过程中，计算burst

重点关注“burst”这个参数，这个参数的取值是0-65535，指的是一次搬运多少个“block”（也就是32字节）到目的地。65535这个数值足够大了，不需要搬运多次了，所以nburst直接取1即可。而“burst”这个参数则需要计算一下：用总传进来的总数据量除以每个block存放多少个元素，再向上取整就能得出需要多少个burst了

1. 分阶段调用计算指令的过程中，计算循环次数

参数表中的move\_offset是内存中数据的偏移量，move\_num则是在这个AI Core中一共要计算多少个数据。至于UB一次能放下多少数据，前边我们已经计算过了，就是“ub\_tensor\_size”，那么计算循环次数的方法就是：

loop\_time = move\_num//self.ub\_tensor\_size

UB一次能放下248KB的数据，也就是最多124KB的A张量，124KB的B张量。而一条vec\_add能计算的数据量也是十分有限的，故也会涉及多次循环计算。

1. 应用双缓冲技术来进一步加速计算性能

主要可以解决数据依赖问题。

我们知道网格调度器通过block内的warp切换隐藏内存访问和计算延迟；而通过block之间的切换隐藏块内同步延迟，但这里由于寄存器和共享内存资源的限制，一个SMX上最多只能部署一个活跃block。因而无法再通过增加每个SMX上可以并发的block数量来进一步提升效率。

类似单个线程内的并行和并发，这时我们就需要考虑如何在单个block内寻找更多的并行和并发机制。容易看出主循环体内的并行和并发受限于块内同步，因此如果消除下一个循环步预取的数据和当前计算中所用数据的依赖关系，则第二个同步原语是可以消除的，为此我们通过双倍的共享内存作为双缓冲来消除这种影响。

使用双缓冲技术将当前计算的读取缓冲区和负责装载下一步数据的缓冲区相分离，从而加载下一步数据到共享内存的操作可以和当前的计算重叠执行，减少了同步开销从而减少流水线的中断。

可能的缺点：但是使用双缓冲时需要注意当前的设备上具体的效果，此外不同的计算规模效果也会不同。如果计算规模比较小，那么使用双缓冲会由于更复杂的地址计算而降低效率。