哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

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

姓名：

学号：

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

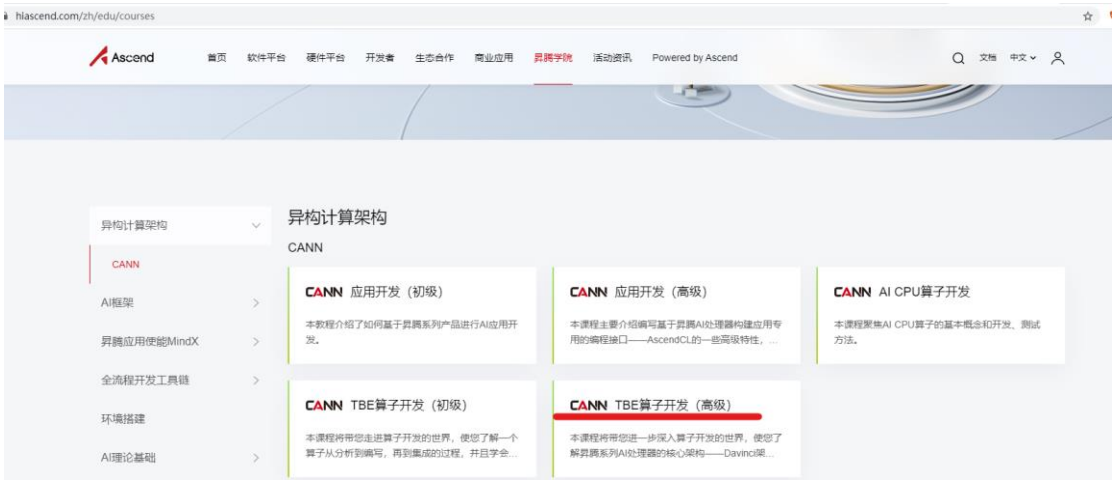

图 1 昇腾学院异构计算架构 CANN 中 TBE 算子开发（高级）

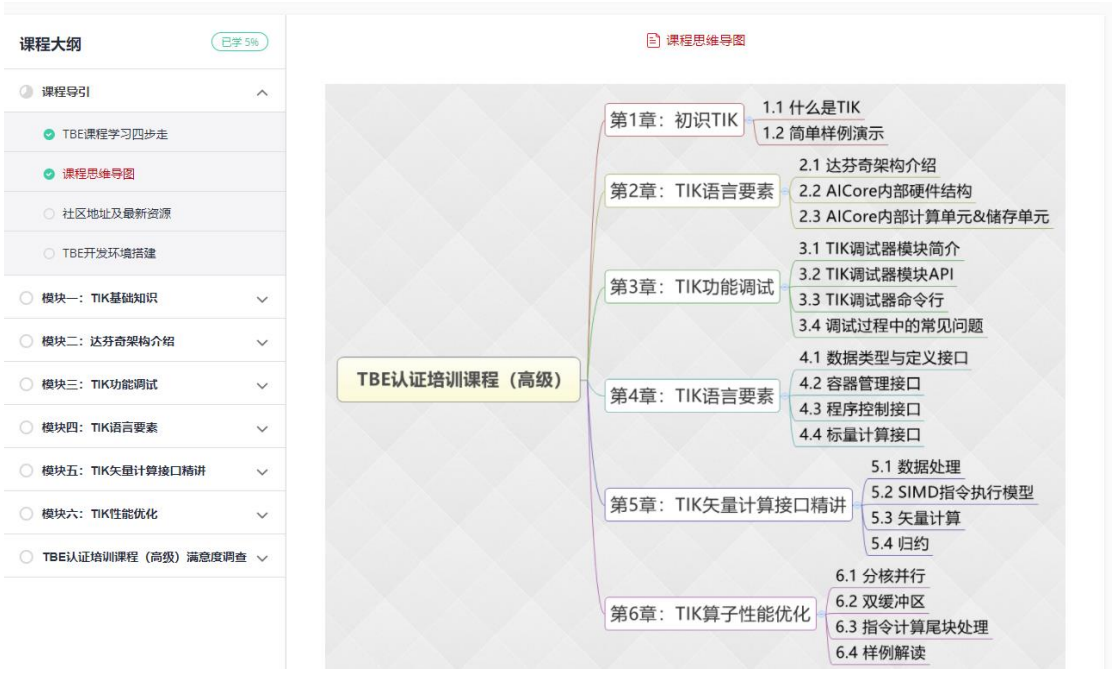

图 2 本次实验主要涉及模块一到模块五

3、 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？（ ）

A、整颗芯片　　B、AI CPU　　C、AI Core　　D、DVPP

答：C

2） 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：深度学习中神经网络所必需的常用计算，例如矩阵相乘，能实现对整形和浮点型数据实现乘加等计算。

AI CPU 适合执行： 不适合在 AI core 上处理的其余任务，如一些不适合转换成并行处理的任务。

3） 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

4） 向量计算单元一次能够处理的数据量为多少？（ ）

A、512 字节 B、256 字节 C、128 字节 D、64 字节

答：B

5） 简述 TIK 算子开发过程

答：

1） 算子分析

算子开发前进行算子分析，明确算子的功能、输入、输出，选取算子代码实现方式，规划算子类型名称以及算子实现函数名称等。

2） 算子实现

算子计算逻辑及调度的实现。

3） 算子编译

编译自定义算子工程，生成自定义算子安装包并进行自定义算子包的安装，将自定义算子部署到算子库（OPP）。

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

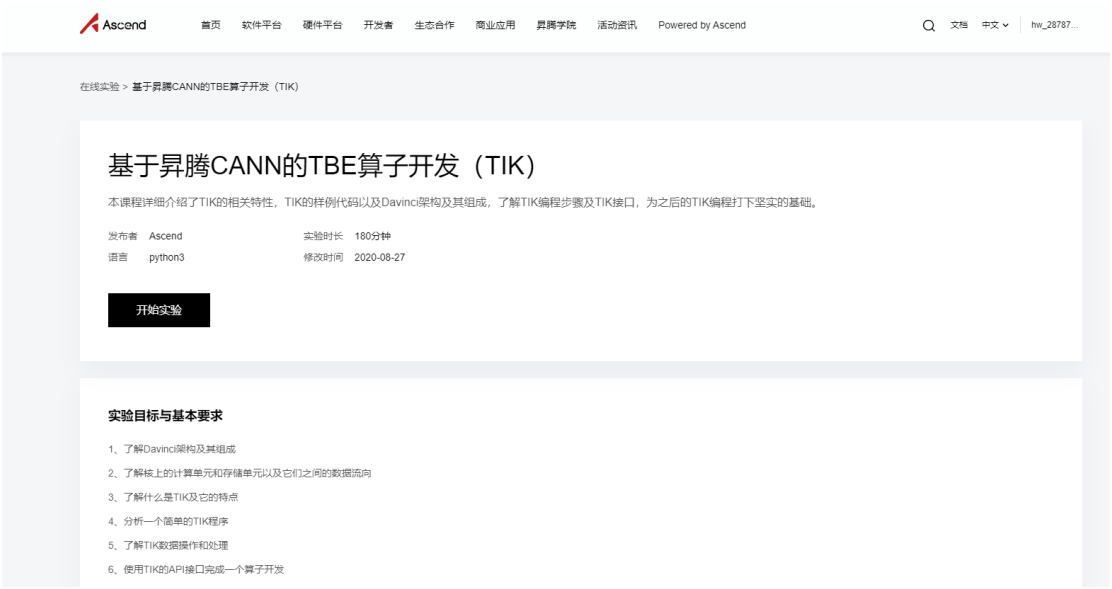

图 3 TIK 算子开发实验

参考文档：

https://support.huaweicloud.com/tbedevg-cann503alpha2training/atlastikapi_07_0001.html

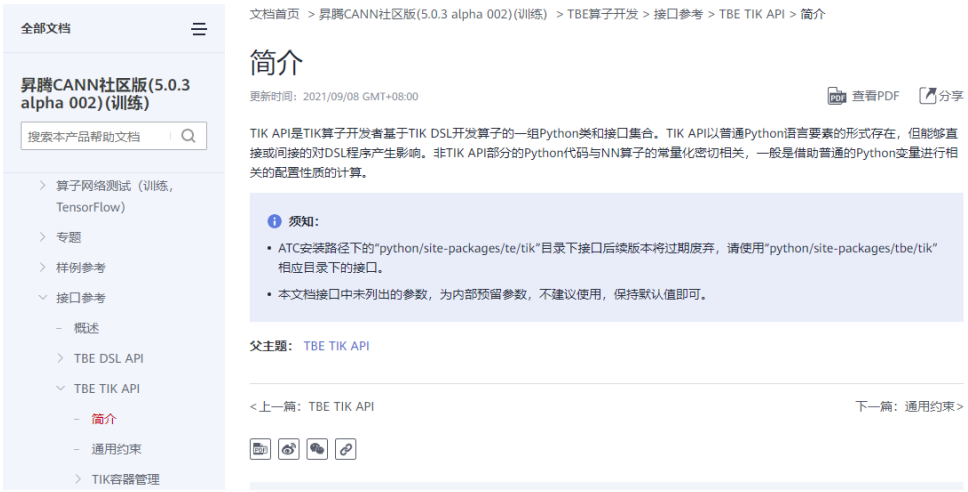

图 4 CANN 官方文档，TBE TIK 相关 API

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：可在 for 循环中开启 double buffer 和多核运行功能，可以加快程序运行的速度。同时可以节省使用的临时变量的大小。

2. 实验代码（30 分）

（粘贴对应的完整代码截图）

```
def element_add_test():
    tik_instance = tik.Tik()
    set_current_compile_soc_info("Ascend310")
    data_A = tik_instance.Tensor("float16", (16, 16, 16, 16, 16), name="data_A", scope=tik.scope_gm)
    data_B = tik_instance.Tensor("float16", (16, 16, 16, 16, 16), name="data_B", scope=tik.scope_gm)
    data_C = tik_instance.Tensor("float16", (16, 16, 16, 16, 16), name="data_C", scope=tik.scope_gm)

    data_a_ub = tik_instance.Tensor("float16", (1, 4, 16, 16, 16), name="data_a_ub", scope=tik.scope_ubuf)
    data_b_ub = tik_instance.Tensor("float16", (1, 4, 16, 16, 16), name="data_b_ub", scope=tik.scope_ubuf)
    data_c_ub = tik_instance.Tensor("float16", (1, 4, 16, 16, 16), name="data_c_ub", scope=tik.scope_ubuf)

    # define other scope_ubuf Tensors
    with tik_instance.for_range(0, 16) as i0:
        with tik_instance.for_range(0, 4) as i1:
            # move data from out to UB
            tik_instance.data_move(data_a_ub,data_A[i0*65536+i1*16384],0,1,1024,0,0)
            tik_instance.data_move(data_b_ub,data_B[i0*65536+i1*16384],0,1,1024,0,0)
            # calculate with TIK API
            tik_instance.vadd(128,data_c_ub,data_a_ub,data_b_ub,128,1,1,1,8,8,8)

            # move data from UB to OUT
            tik_instance.data_move(data_C[i0*65536+i1*16384],data_c_ub,0,1,1024,0,0)

    tik_instance.BuildCCE(kernel_name="element_add_test", inputs=[data_A, data_B], outputs=[data_C])

    return tik_instance
```

（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）

```
In [31]: !bash ../deploy_env/eltwise.sh

[DEBUG] config_file.cc:109 assignConfigPath Using config file [/home/HwHiAiUser/Ascend/
ascend-toolkit/5.0.2.alpha005/x86_64-linux/toolkit/tools/simulator/Ascend310/lib/config
_pv_aicore_model.toml]
loc: 0 src: 12.484 exp: 12.484
loc: 1 src: 10.28 exp: 10.28
loc: 2 src: 12.84 exp: 12.84
loc: 3 src: 6.285 exp: 6.285
loc: 4 src: 14.53 exp: 14.53
Is allclose: True
Total Num: 1048576 error cnt: 0 error percent: 0.0
compare success
resubmiting your score...
upload scores success.
...
reference score: 100
```

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. 张量 shape、dtype、数据排布格式等信息

   在商用算子开发的时候要能够适应任何合法的数据类型、数据形状、数据排布格式，此时这些信息就要从算子的方法声明中获取。而在初始化的时候可以使用张量的 get('shape')等方法获得这些信息。

2. UB 可用空间

商用算子开发的时候是希望能运行在多种型号的昇腾 AI 的处理器上的，但是在不同型号的昇腾 AI 处理器上 UB 的可用空间是不同的，整个 UB 通常被划分为两个部分，一部分用于 Vector 计算，一部分用于 Scalar 计算。可以使用 tbe_platform.get_soc_spec("UB_SIZE")获取一下 Vector 计算单元可用的 UB 空间大小

3. 数据在内存和 AI Core 之间如何传递

既然要用到两个 AI Core 做计算，内存中的数据自然也会比分成两份。每一份的大小是“data_num_each_core”。这里我们给在每个 AI Core 上做计算的方法起个名字为“vadd_compute_each_core”。

那么这个“vadd_compute_each_core”方法想来应该传入的是在每个 AI Core 上取内存中数据的起始地址，以及想在这个 AI Core 上做计算的总数据量。当循环变量“index”为 0 时，起始地址就是 0*vadd_compute_each_core；当“index”为 1 时，起始地址就是 1*vadd_compute_each_core，以此类推