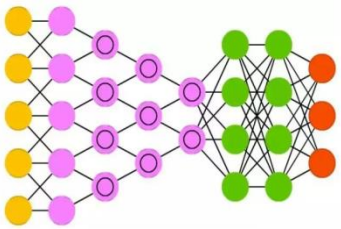
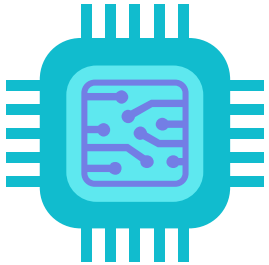
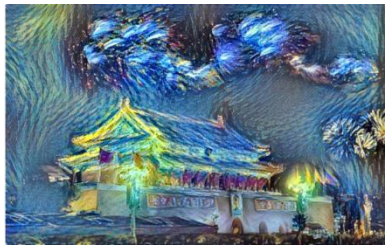
智能计算系统

实验一 基于BANG语言的算子开发

中国科学院计算技术研究所

Driving Example

输入 ▶ 建模 ▶ 实现 ▶ 运行 ▶ 输出

神经网络　深度学习　编程框架　Bang　平台架构　实验一　实验二

实验一将基于BANG 语言完成面向寒武纪系列开发平台的高性能算子实现

提纲

- 实验目的与要求
- 背景介绍
- 实验步骤
- 演示运行

实验目的

- 使用智能编程语言进行算子开发
- 对高性能算子库进行扩展
- 将算子集成到TensorFlow中

实验要求

- 必做题：实现风格迁移相关算子PowerDifference
- 选做题：以下算子4选1
  - Softmax
  - Cosine相似度
  - Batch Normalization
  - Triplet Loss

提纲

- 实验目的与要求
- 背景介绍
- 实验步骤
- 演示运行

- PowerDifference 介绍

$$PowerDifference = (X - Y)^Z$$

- X和 Y 是张量数据类型， Z 是标量数据类型

编译器（CNCC）

- 将使用智能编程语言编写的程序编译成 DLP 底层指令
- CNCC编译选项

常用选项	说明
-E	编译器只执行预处理的步骤，生成预处理文件
-S	编译器只执行预处理、编译的步骤，生成 MLISA 汇编文件
-c	编译器只执行预处理、编译、汇编的步骤，生成 ELF 格式的汇编文件
-o	将输出写入到指定的文件
-x	为输入的文件指定编程语言，e.g.:bang 等
–target=	指定生成 DLP 端的目标文件的格式，该文件和目标 HOST 平台的目标文件一起链接成目标 HOST 端的可执行程序，所以其值为目标 HOST 端二进制文件格式，e.g: x86_64, armv7a 等
–bang-mlu-arch=	为输入的 BANG 程序指定 DLP 的架构，e.g.: DLP270 等
–bang-stack-on-ldram	栈是否放在 LDRAM 上，默认放在 NRAM 上，如果该选项开启，栈会放在 LDRAM 上
–cnas-path=	指定汇编器的路径

调试器（CNGDB）

- 调试器是面向智能编程语言程序的调试工具，能够支持搭载 DLP 硬件的异构平台调试
- 同时支持 Host 端 C/C++ 代码和 Device 端 BCL 的调试，同时两者调试过程的切换对于用户而言也是透明的
- 主要用法：
  - 在 xxx.mlu 文件 x 行插入断点:break xxx.mlu : x (break 可以缩写为 b)
  - 执行程序:run(run 可以缩写为 r)
  - 继续执行至下一个断点:continue(continue 可以缩写为 c)
  - 删除断点 1:delete 1(delete 可以缩写为 d)
  - 打印变量 xx:print xx(print 可以缩写为 p)
  - 打印内存:x/个数 + 字长 (x 是 examine 的缩写)

提纲

- 实验目的与要求
- 背景介绍
- 实验步骤
- 演示运行

1、算子实现

- 在设备端，基于Bang语言实现PowerDifference算子

```
__mlu_entry__ void PowerDifferenceKernel(half* input1, half* input2, int pow, half*
    output, int len)
```

- 其中，input1对应向量X，input2对应向量Y，pow对应标量Z

$$PowerDifference = (X - Y)^Z$$

2、算子测试

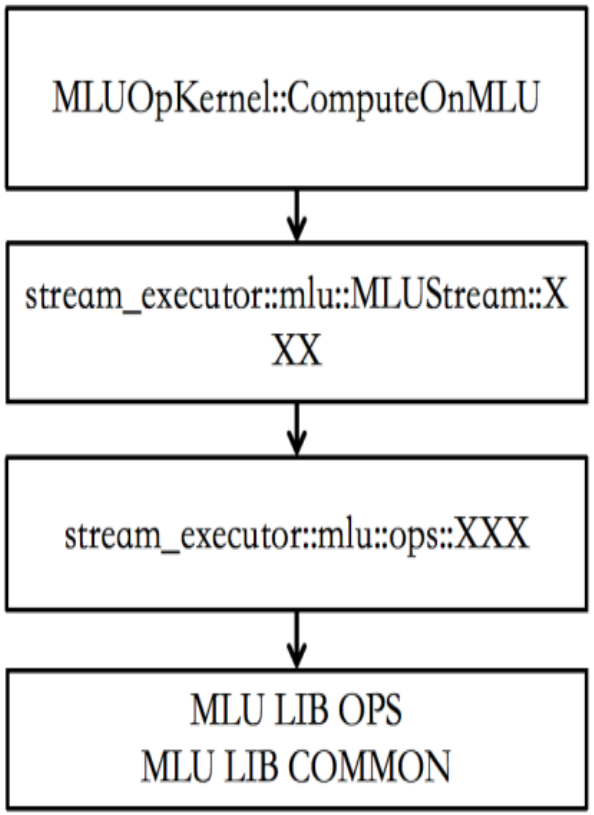
- 具体包括：编写运行时程序，编写 main 程序和编译运行等步骤
- 1）运行时程序powerDiff.cpp
  - 在 MLUPowerDifferenceOp 中通过一系列 CNRT 接口调用 BCL 算子的具体实现（PowerDifferenceKernel）
- 2）主程序main.cpp
  - 调用 MLUPowerDifferenceOp 函数对输入数据进行计算，并将结果输出到文件中
- 3）编译运行，生成power_diff_test可执行程序

3、框架集成

- 1) PluginOp接口封装
- 声明算子参数初始化函数CreateOpParam
  - 完成 static tensor 的创建、 CPU相关的赋值，再执行 cnmlBindConstData 绑定常量数据
- 声明算子参数销毁函数 DestroyOpParam
  - 销毁 statictensor 和影响 CPU 的指针，然后销毁为 cnmlPluginXXXOpParam 分配的地址空间
- 声明算子构建接口 Create
  - 通过调用 cnmlCreatePluginOp 传递 BCL 算子函数指针、输入和输出变量指针完成算子创建。创建成功后可以得到 cnmlBaseOp_t 类型的指针
- 声明单算子运行接口 Compute
  - 通过调用 cnmlComputePluginOpForward 利用前面创建的 cnmlBaseOp_t 的指针和输入输出变量指针完成上述计算过程

3、框架集成

- 2）DLP算子集成到TensorFlow框架
  - MLUOpKernel：继承 TensorFlow 中的 OpKernel 类，作为与TensorFlow算子层的接口
  - MLUStream：与 MLUOpKernel 类接口关联，负责 MLU算子的实例化并管理算子缓存
  - MLUOps：负责 TensorFlow 算子的 DLP 实现，可以是单算子也可以是内存拼接的算子
  - MLULib：对 CNML 和 CNRT 接口的直接封装，只包含极少的 TensorFlow 数据结构
- 集成的流程包括：算子注册、定义 MLULib 层接口、定义 MLUOps 层接口、定义 MLUStream 层接口以及定义 MLUOpKernel 层接口



3、框架集成

- 3）算子测试
  - 在新增自定义的 PowerDifference 算子与 TensorFlow 框架的集成完成后，用户需要使用Bazel重新编译 TensorFlow，再使用 Python 侧的 API 对新集成的算子功能进行测试
  - 针对相同的输入数据，分别使用Bang算子、CPU上已经实现好的powerDifference算子，计算二者的平均错误率MAE

提纲

- 实验目的与要求
- 背景介绍
- 实验步骤
- 演示运行

演示运行

谢谢大家！