智能计算系统
实验第五章
智能编程语言

张欣

中国科学院计算技术研究所

目录

实验目的

- 掌握使用智能编程语言BANG进行算子开发、编译扩展高性能库算子，并集成到Pytorch框架中的方法和流程。能够用BANG实现 Sigmoid算子，并集成进Pytorch 的推断网络高效地运行在DLP 硬件上。

BANG编程流程

- 智能编程语言采用异构混合编程和编译。
- 一个完整的程序包括主机端（Host）程序和设备端（Device）程序。
- 主机端程序主要调用运行时库接口执行内存申请、释放、拷贝，Kernel的控制执行；
- 设备端程序使用BANG特定的语法规则执行计算部分和并行任务。
- 用户可以在主机端输入数据，做一定处理后，通过一个Kernel启动函数将相应输入数据传给设备端，设备端进行计算后，再将计算结果拷回主机端。

编译器（CNCC）

- CNCC是将使用智能编程语言（BCL）编写的程序编译成DLP底层指令的编译器。为了填补高层智能编程语言和底层DLP硬件指令间的鸿沟，DLP的编译器通过复杂寄存器分配、地址空间推断、全局指令调度等技术实现编译优化，以提升生成的二进制指令性能。

- 开发者使用BCL开发自己的DLP端的源代码，首先通过前端CNCC编译为汇编代码，然后汇编代码由CNAS汇编器生成DLP上运行的二进制机器码。

高性能算子库（CNNL、MLU-OPS）

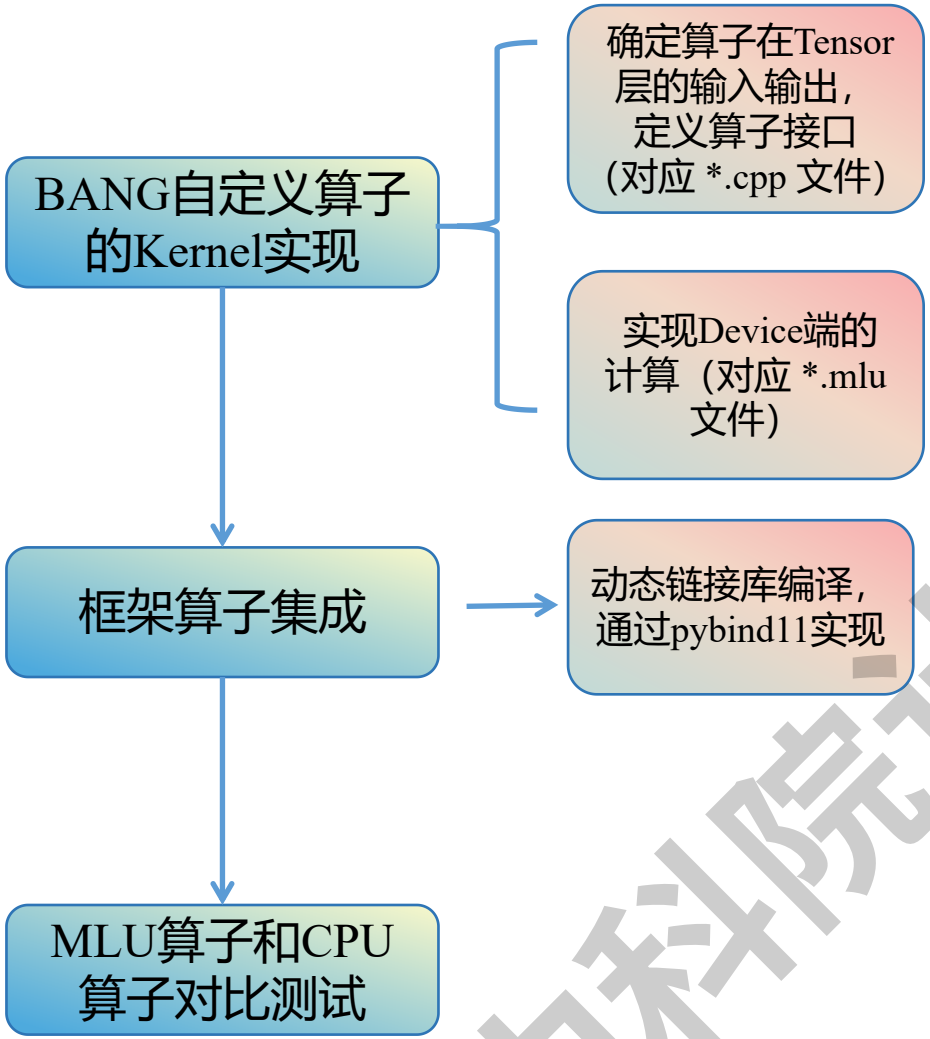
- CNNL高性能算子库是基于BANG开发的算子库集合，CNNL为PyTorch、TensorFlow、PaddlePaddle等开源框架提供了运行在MLU硬件的完备算子集合，用户无需用BANG开发即可通过CNNL运行主流网络的训练和推理并获得最优性能。
- MLU-OPS是CNNL算子库的开源版本，提供基于MLU使用C接口或者Python接口开发高性能算子的示例代码。MLU-OPS旨在通过提供示例代码，供开发者参考使用，可用于开发自定义算子，实现对应模型的计算。

实验内容

用智能编程语言BANG来实现自定义算子 Sigmoid 的计算逻辑（Kernel函数），通过 PyTorch 的自定义算子扩展机制（ MLUExtension参考CUDAExtension），将自定义算子 Sigmoid 集成到编程框架 PyTorch 中。

BangC开发手册可参考：
https://www.cambricon.com/docs/sdk_1.12.0/cntoolkit_3.4.2/cambricon_bang_c_4.4.0/index.html

实验步骤




```
# Sigmoid自定义算子实验根目录
├── README.md: 描述算子功能的说明文档
├── mlu_custom_ext: 生成的module模块用于在python层导入。
│   ├── __init__.py: python包固有文件
│   ├── mlu: mlu代码文件，根据实际情况自己创建，在setup.py中修改即可（建议使用目录管理BANG代码）。
│   │   ├── include: 头文件目录（头文件和实现分离，属于代码习惯，建议采用此布局）
│   │   │   ├── bang_sigmoid_sample.h: 实现对mlu函数的封装。
│   │   │   ├── kernel.h: BANG代码中的宏，良好的组织代码的需要。
│   │   │   └── custom_ops.h: 算子对外头文件。
│   │   └── src
│   │       ├── bang_sigmoid.cpp: 对PyTorch层面Tensor的封装，和自定义算子中xxx_internal的实现类似。
│   │       └── bang_sigmoid_sample.mlu: 核心BangC实现。
│   └── mlu_functions: 合理的组织自己的代码方便后续调用。
│       ├── __init__.py: 包必备文件。
│       └── mlu_functions.py: 对C++代码的封装。
├── setup.py: 构建包的脚本。
└── tests
    └── test_sigmoid.py: 对绑定代码的python侧测试。
```

实验代码目录结构

实验步骤

Sigmoid 函数释义

文件名	函数名	释义
test_sigmoid.py	test_forward_with_shape()	pytest 测试函数
test_sigmoid.py	test_backward_with_shape()	pytest 测试函数
mlu_functions.py	forward()	继承 torch.autograd.function 类的正向 sigmoid 接口
mlu_functions.py	backward()	继承 torch.autograd.function 类的反向 sigmoid 接口
bang_sigmoid.cpp	active_sigmoid_mlu()	C++ 函数接口，属于 torch_mlu 命名空间，操作的是 PyTorch 的 Tensor，被 pybind11 封装进 libmlu_custom_ext 库
bang_sigmoid_sample.mlu	bang_sigmoid_kernel_entry()	BANG 编程中主机端的 C++ 函数入口，被 Pytorch 的 C++ 接口调用
bang_sigmoid_sample.mlu	bang_sigmoid_kernel()	BANG 编程中设备端核函数，被主机端程序使用 «<<>> 语法调用
bang_sigmoid_sample.mlu	bang_sigmoid_sample()	C++ 测试用例封装的函数接口，仅供 C++ 测试使用，PyTorch 自定义算子中并未调用

（1）在根目录下执行python setup.py install，完成定义算子的编译和安装。
（2）进入tests目录，执行python test_sigmoid.py完成精度测试。
（3）执行python test_sigmoid.py进行性能测试。

评估标准

本实验主要关注 BANG C 自定义算子的实现与验证、与框架的集成以及完整的模型推断。模型推断的性能和精度应同时作为主要参考指标。因此，本实验的评估标准设定如下：

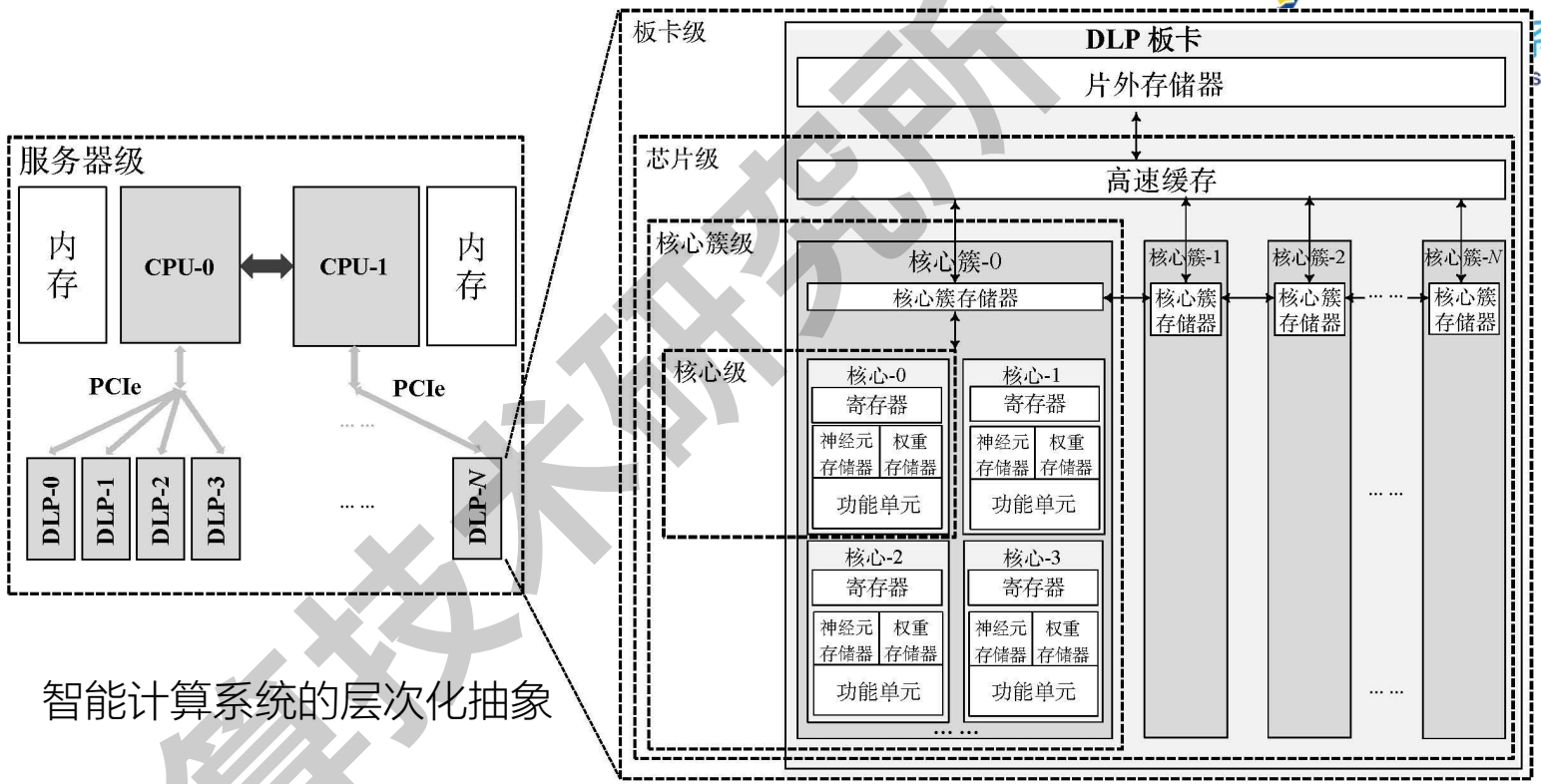
- 60 分标准：实现 Sigmoid 主程序、核函数、pybind 接口，使用 setuptools 编译并安装成功；
- 80 分标准：在 60 分基础上，完成精度对比测试用例，使用 numpy 的 assert_array_almost_equal 接口评估精度误差在 decimal=3 以内。
- 100 分标准：在 80 分基础上，完成性能测试，前向 Sigmoid 测试耗时小于 25ms，反向 Sigmoid 测试耗时小于 70ms。

目录

实验目的

- 掌握使用智能编程语言优化算法性能的原理，掌握智能编程语言的调试和调优方法，能够使用智能编程语言在MLU上加速矩阵乘的计算。

BCL编程模型



智能计算系统的层次化抽象

- 从服务器级依次到处理器核级，存储单元的数据访问延迟依次递减，数据访问带宽依次递增，存储单元的空间大小依次递减。
- 具体编程时，需要调用BCL的memcpy函数，手动完成数据在Global Memory（GDRAM），SRAM，NRAM，WRAM之间的调度。
- 在编程实现时，如果需要将数据从GDRAM拷贝到SRAM，只需调用智能编程语言中的Memcpy函数，同时指定拷贝方向为GDRAM2SRAM。
- 在编程时可以在程序中指定运行一次任务调用的计算资源数量。特别地，我们称一次执行只调用一个Core的任务为BLOCK任务。一次执行只调用一个Cluster的任务为UNION1任务，对应调用两个Cluster与四个Cluster的任务分别为UNION2和UNION4。

DLP并行编程

在BCL中使用一系列的并行内嵌变量来帮助使用者充分发挥DLP的并行特征。

- Core变量: coreDim、coreId
- Cluster变量：clusterDim、clusterId
- Task变量：taskDim、taskId

Notifier接口

- Notifier是一种轻量级任务，不像计算任务那样占用计算资源，而是通过驱动从硬件读取一些运行参数，只占用很少的执行时间（几乎可以忽略不计）。Notifier可以像计算任务任务一样放入Queue（队列）中执行，在队列中均遵循FIFO（先进先出）调度原则。可以使用Notifier来统计Kernel计算任务的硬件执行时间。

实验内容

本节实验对于矩阵乘运算，首先使用BANG C语言实现一个标量版本，然后利用NRAM存储优化、向量化、多核并行优化、三级流水优化、五级流水优化，逐步实现一个高性能矩阵乘，每一个优化步骤都除了和CPU版本做性能对比外，还和上一步的BANG C优化做对比，从而逐步理解智能编程语言的优化技巧。

(1)实现一个CPU版本的标量矩阵乘，将标量矩阵乘实现迁移到MLU上，做性能对比；

(2)利用BANG C的NRAM地址空间加速标量版本的矩阵乘，对比性能提升；

(3)利用BANG C提供的向量化接口__bang_matmul加速矩阵乘，对比性能提升；

(4)利用BANG C提供的Block任务，使用MLU的多核做并行加速，对比性能提升；

(5)利用BANG C提供的异步拷贝接口__memcpy_async做三级流水优化，对比性能提升；

(6)利用BANG C提供的Union任务和SRAM地址空间，做五级流水优化，对比性能提升；

矩阵乘标量实现
——01_scalar.mlu

实现较小规模的矩阵乘（MxNxK=128x256x128），分别在CPU和MLU（Machine Learning Unit）上进行矩阵乘法计算，并对结果进行验证。

矩阵乘标量NRAM实现
——02_scalar_nram.mlu

本实验步骤目的是利用BANG C的NRAM地址空间来加速标量版本的矩阵乘，原理是通过BANG C的静态数组在NRAM上为左右矩阵和结果矩阵声明空间，将小规模的矩阵一次性拷贝至NRAM，在NRAM上运算完成后一次性拷出到GDRAM空间，由于标量读写NRAM空间的性能远高于GDRAM空间，所以性能会有几十倍的提升。

矩阵乘向量NRAM实现
——03_vector_nram.mlu

MLU 硬件架构支持 SIMD 指令集，所以为了发挥算力优势，必须调用 BANG C 封装的 Builtin 函数 __bang_matmul 进行向量化加速。本实验步骤仍然用较小规模的单核矩阵乘，仅仅将标量循环实现替换为调用 __bang_matmul 函数实现，性能将有几千倍量级的提升。

矩阵乘多核向量NRAM实现
——04_vector_nram.mlu

本实验实现大规模的矩阵乘（MxNxK=524288x256x128），步骤是利用多核加速的原理是将超大规模的矩阵乘，在M维度上拆分为BLOCKS个可并行的小矩阵乘，然后利用BANGC提供的taskId、taskDim等并行变量将核函数改写为并行版本，改写后相比CPU的标量单核版本基线要快几千至上万倍。

矩阵乘多核向量NRAM三级流水实现
——05_vector_nram_blocks_pipe3.mlu

整体逻辑可以分为三部分：Load，Compute 和 Store。为了避免 Load 和 Store 产生冲突，需要引入两块独立的 Ping-Pong 缓冲区。由于 NRAM 大小有限，要进行多次普通的向量计算过程如图中绿色方框所示，顺序执行 0->1->2，即 [L0-C0-S0]->[L1-C1-S1]-> [L2-C2-S2] 。如果通过软流水技术重新排列，则变成橘红色方框格式，顺序执行 a->b->c，即 [S0-C1-L2]->[S1-C2-L3]->[S2-C3-L4]，可以写成新的循环形式。通过以上方式有效地隐藏了数据传输的延迟，提高了资源利用率，减少了流水线停顿，从而显著提升了计算效率。

矩阵乘多核向量SRAM五级流水实现
——06_vector_sram_unions_pipe5.mlu

通过引入SRAM和更复杂的五级流水线机制，优化了数据传输和计算的并行处理，大大提高了计算效率。

实验步骤

1. 执行source env.sh进行环境激活
2. 执行bash test.sh进行性能测试

性能对比

测试环境参数如下：CPU：Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz MLU: MLU370-X8@1000MHz MLU Driver: v5.10.26 MLU Firmware: v1.1.6 CNToolkit: v3.9.0

实验步骤	矩阵乘规模 (MxNxK)	CPU 耗时 (ms)	MLU 耗时 (ms)
标量实现	128x256x128	5.857	2716.140
标量 NRAM 实现	128x256x128	5.847	126.067
向量 NRAM 实现	128x256x128	5.904	0.050
多核向量 NRAM 实现	524288x256x128	26075.229	4.563
多核向量 NRAM 三级流水实现	524288x256x128	26076.031	3.824
多核向量 SRAM 五级流水实现	524288x256x128	26081.332	4.073

评估标准

本实验设定的评估标准如下，每升高一级标准，不但要实现当前分值标准的代码，还要实现上一级分值标准的代码，即 100 分标准要求实现“标量 NRAM 实现”、“向量 NRAM 实现”、“多核向量 NRAM 实现”、“多核向量 NRAM 三级流水实现”、“多核向量 SRAM 五级流水实现”共 5 个版本的 BANG C 代码。

- 60 分标准：MxNxK = 128x256x128 规模，实现标量 NRAM 矩阵乘，MLU 计算结果与 CPU 计算结果误差为 0。
- 70 分标准：MxNxK = 128x256x128 规模，实现向量 NRAM 矩阵乘，精度达标，MLU 性能为 CPU 的 xxx 倍以上。
- 80 分标准：MxNxK = 524288x256x128 规模，实现多核向量 NRAM 矩阵乘，精度达标，MLU 性能为 CPU 的 xxx 倍以上。
- 90 分标准：MxNxK = 524288x256x128 规模，实现多核向量 NRAM 三级流水矩阵乘，精度达标，MLU 性能为 CPU 的 xxx 倍以上。
- 100 分标准：MxNxK = 524288x256x128 规模，实现多核向量 NRAM 五级流水矩阵乘，精度达标，性能和三级流水在同一个数量级。

敬请指正！

课程官网：http://novel.ict.ac.cn/aics

MOOC网址：

https://space.bilibili.com/494117284/video