CANN ops-math仓开源

https://gitcode.com/cann

CANN

ops-math仓基础介绍

ops-math是CANN（Compute Architecture for Neural Networks）算子库中提供数学类计算的基础算子库，包括math类、conversion类、random类等算子

- **math类**：基础数学计算类算子，例如Abs、Acos、Add等;
- **conversion类**：张量变换类算子，例如Transpose、Pad、Slice等;
- **random类**：随机数生成类算子，例如Randperm等;

https://gitcode.com/cann

CANN

ops-math仓目录结构介绍

```
├── build.sh                  # 项目工程编译脚本
├── cmake                     # 项目工程编译目录
├── CMakeLists.txt
├── common                     # 项目公共头文件和公共源码
├── docs                      # 项目文档介绍
├── example                   # 使用通用算子开发和调用示例
├── conversion                # conversion类算子
├── ...
├── math                      # math类算子
│   ├── abs                   # abs算子所有交付件，如Tiling、Kernel等
│   │   ├── CMakeLists.txt        # 算子编译配置文件
│   │   ├── docs                  # 算子说明文档
│   │   ├── examples              # 算子使用示例
│   │   ├── op_graph              # 算子构图相关目录
│   │   ├── op_host               # 算子信息库、Tiling、InferShape相关实现目录
│   │   │   └── op_api            # 算子aclnn接口实现目录
│   │   ├── op_kernel             # 算子kernel目录
│   │   └── README.md             # 算子说明文档
│   ├── ...
│   └── CMakeLists.txt            # 算子编译配置文件
├── tests                         # 测试工程目录
├── README.md
├── requirements.txt          # 本项目需要的第三方依赖包
└── scripts                   # 脚本目录，包含自定义算子、kernel构建相关配置文件
```

- **Docs**：包含aclnn接口介绍等md文档；
- **Examples**：算子调用示例；
- **op_graph**：算子图模式场景交付件，例如InferDatatype等；
- **op_host**：算子基础host侧交付件，例如Tiling、aclnn接口等；
- **op_kernel**：算子device侧交付件，包含算子核心实现。

CANN

算子基本开发流程介绍

基于开源仓开发新算子，核心交付件包含Tilling与Kernel两部分内容

1. 前提条件：

   ① 环境部署：开发算子前，请确保基础环境已安装，例如依赖的驱动、固件、CANN软件包等。

   ② 算子设计：分析实际业务诉求，合理设计算子规格，包括算子输入、输出、属性的数据类型、shape等。

2. 工程创建：开发算子前，需按要求创建算子目录，方便后续算子的编译和部署。

3. Tiling实现：实现Host侧算子Tiling函数。

4. Kernel实现：实现Device侧算子核函数。

5. aclnn适配：自定义算子推荐aclnn接口调用，需完成二进制发布。如需入图，请参考附录。

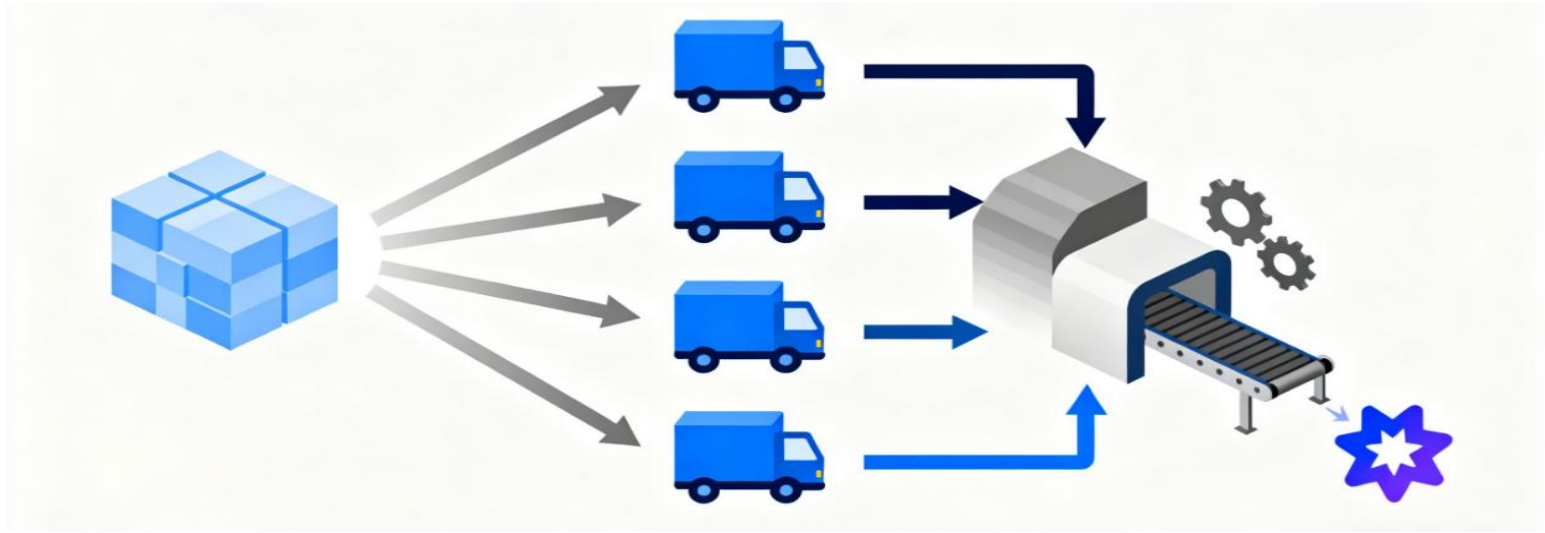
6. 编译部署：通过工程编译脚本完成自定义算子的编译和安装。

7. 算子验证：通过常见算子调用方式，验证自定义算子功能。

- **Host侧Tiling实现：**
  由于NPU中AI Core内部存储无法完全容纳算子输入输出的所有数据，需要将数据分批次进行处理。切分数据的算法称为Tiling算法或者Tiling策略。
- **Device侧Kernel实现：**
  Kernel实现即算子核函数实现，通过调用计算、数据搬运、内存管理、任务同步API，实现算子逻辑。其核心逻辑基本上都为计算密集型任务，需要在NPU上执行。

CANN

Tiling策略：找出输入转化为输出的最短时间



任务分配

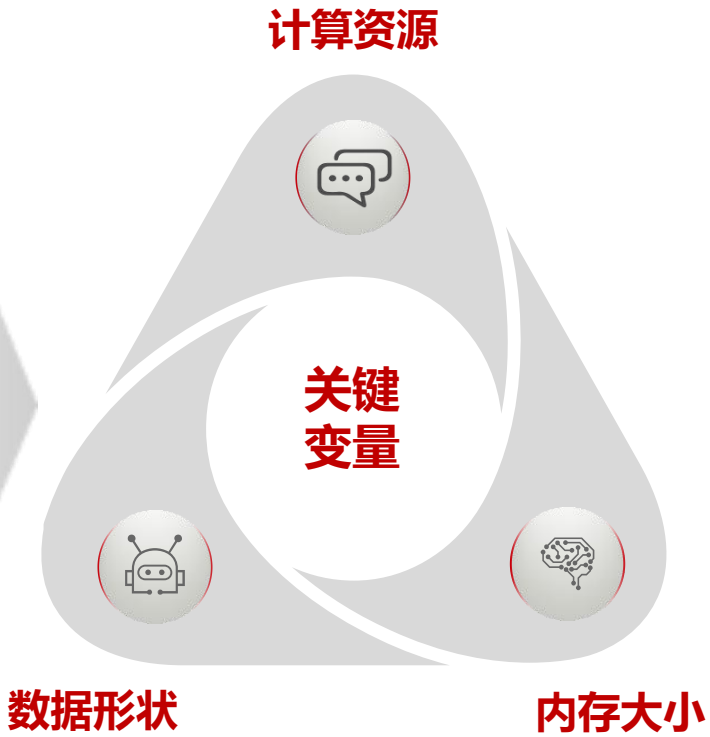
分配计算子任务

任务装载

装载任务到NPU

任务输出

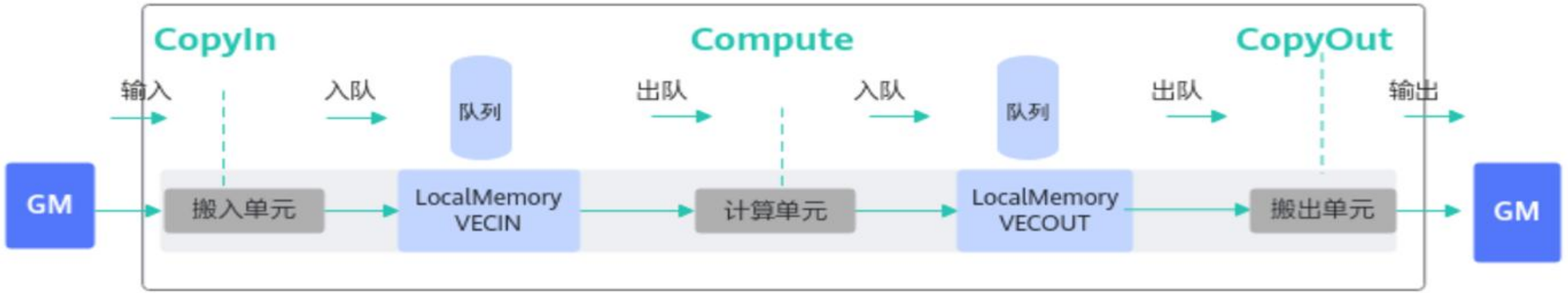
NPU计算后输出



CANN

Kernel核函数性能优化分析

根据计算密集型（Compute-bound）和访存密集型（Memory-bound）分析性能瓶颈



计算密集型

优化：
- 1. 简化计算逻辑
- 2. 流水并行

访存密集型

优化：
- 1. 计算换带宽

CANN

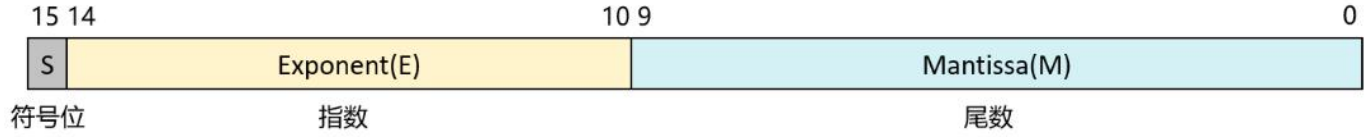
ops-math仓算子开发

ops-math仓算子开发-IsInf算子

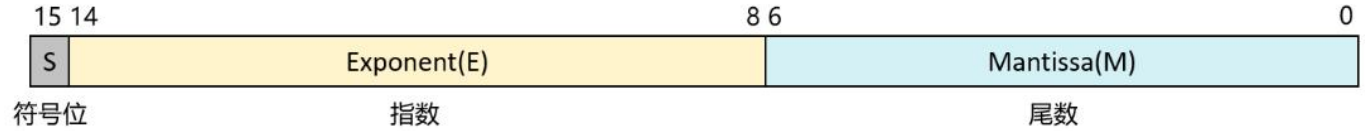
IsInf算子功能：判断张量中哪些元素是无限大值，即是±inf。

FP32的二进制结构包含1bit符号位、8bit指数位和23bit尾数位，简写为**E8M23**。

FP16的二进制结构包含1bit符号位、5bit指数位和10bit尾数位，简写为**E5M10**。



BF16的二进制结构包含1bit符号位、8bit指数位和7bit尾数位，简写为**E8M7**。



当E全为1，M全为0时，表示的结果为±inf。

实现原理：

无限大数的二进制表示为：指数位全1且尾数位全0

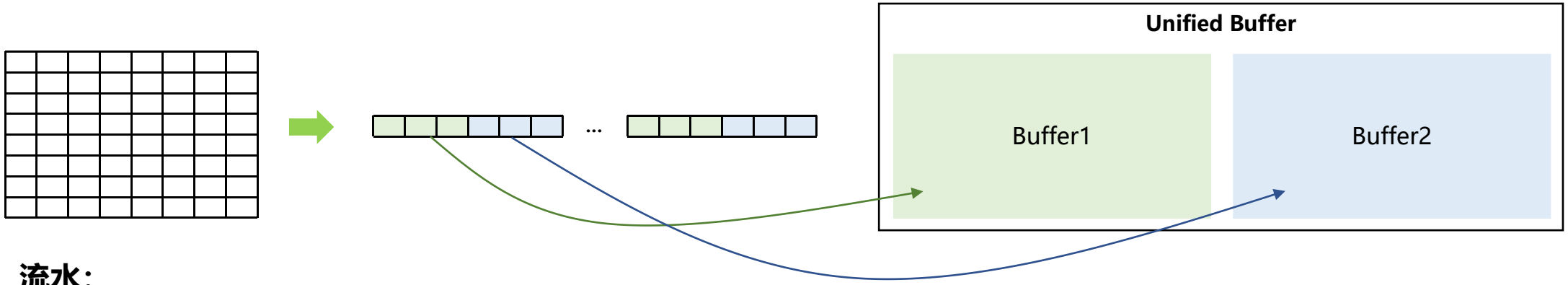
需要判断指数位全1，且尾数位全0

CANN

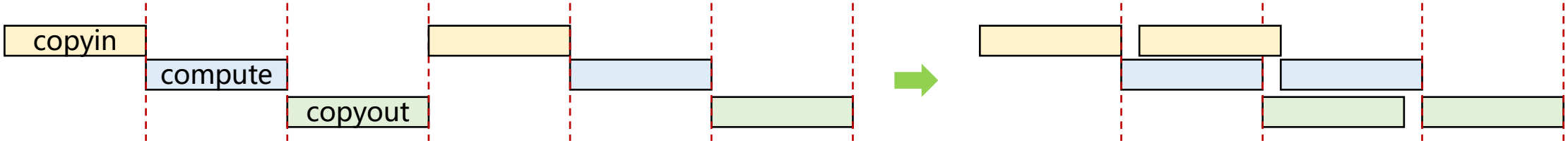
ops-math仓算子开发-IsInf算子Tiling设计

Tiling设计:

所有shape进行合轴，按照UB大小切分最大搬运数据块，同时给DoubleBuffer留两倍空间。



流水:



分核设计:

CANN

ops-math仓算子开发-IsInf算子Kernel设计

Kernel设计 （以FP16为例）：

设计逻辑：

整数减去指数位mask

去掉符号位

判断得到的值是否为0

ops-math仓算子开发-IsInf算子原型定义

op_host及op_kernel目录结构

```
├── op_host
│   ├── config
│   │   ├── ascend310p
│   │   ├── ascend910_93
│   │   └── ascend910b
│   │       ├── is_inf_binary.json
│   │       └── is_inf_simplified_key.ini
│   ├── is_inf_def.cpp
│   ├── is_inf_infershape.cpp
│   ├── is_inf_tiling_arch35.cpp
│   ├── is_inf_tiling_arch35.h
│   ├── is_inf_tiling_def.h
│   ├── is_inf_tiling.cpp
│   └── CMakeLists.txt
└── op_kernel
    ├── arch35
    ├── is_inf.cpp
    └── is_inf.h
```

https://gitcode.com/cann

is_inf_def.cpp 算子原型定义

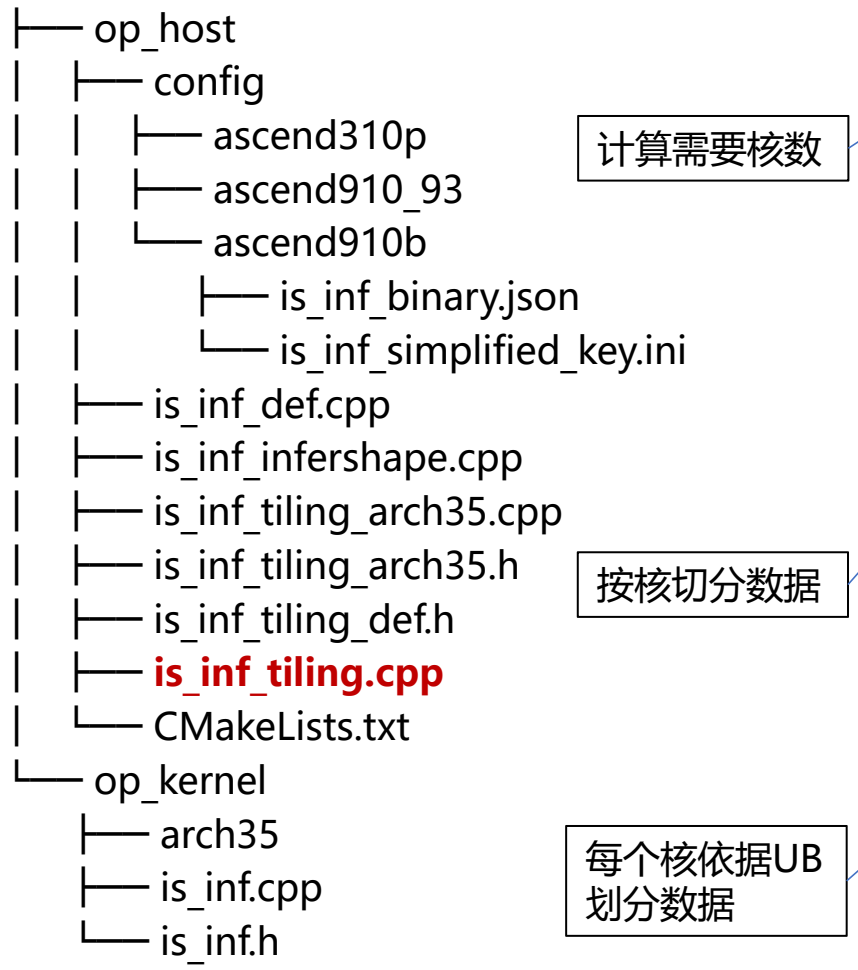
```cpp
namespace ops {
class IsInf : public OpDef {
public:
    explicit IsInf(const char* name) : OpDef(name)
    {
        this->Input("x")
            .ParamType(REQUIRED)
            .DataType({ge::DT_FLOAT16, ge::DT_FLOAT, ge::DT_BF16})
            .Format({ge::FORMAT_ND, ge::FORMAT_ND, ge::FORMAT_ND})
            .UnknownShapeFormat({ge::FORMAT_ND, ge::FORMAT_ND, ge::FORMAT_ND})
            .AutoContiguous();
        this->Output("y")
            .ParamType(REQUIRED)
            .DataType({ge::DT_BOOL, ge::DT_BOOL, ge::DT_BOOL})
            .Format({ge::FORMAT_ND, ge::FORMAT_ND, ge::FORMAT_ND})
            .UnknownShapeFormat({ge::FORMAT_ND, ge::FORMAT_ND, ge::FORMAT_ND})
            .AutoContiguous();

        this->AICore().AddConfig("ascend910b");
        this->AICore().AddConfig("ascend910_93");
    }
};
OP_ADD(IsInf);
```

CANN

ops-math仓算子开发-IsInf算子tiling实现

op_host及op_kernel目录结构



计算需要核数

按核切分数据

每个核依据UB划分数据

is_inf_tiling.cpp tiling切分逻辑

```cpp
uint32_t IsInfTiling::GetNeedCoreNum(uint32_t maxCoreNum) const
{
    uint32_t coreNum = (uint32_t)CeilAlign(totalData, DATA_BLOCK);
    if (coreNum < maxCoreNum) {
        return coreNum;
    } else {
        return maxCoreNum;
    }
}

void IsInfTiling::AssignDataToEachCore()
{
    perCoreData = totalData / needCoreNum;
    perCoreData = perCoreData / DATA_BLOCK * DATA_BLOCK;
    uint32_t tailData = totalData - perCoreData * needCoreNum;
    tailDataCoreNum = tailData / DATA_BLOCK;
    lastCoreData = perCoreData + tailData % DATA_BLOCK;
}

uint32_t IsInfTiling::GetUsableUbMemory(uint64_t ubSize)
{
    // The remaining UB size is split in two, double buffer, input and output
    uint32_t usedUbSize = uint32_t(ubSize - resevedUbSize -
tilingData.GetDataSize()) / UB_PART;
    usedUbSize = usedUbSize / dataBlockSize * dataBlockSize;
    return usedUbSize;
}
```

ops-math仓算子开发-IsInf算子kernel实现

op_host及op_kernel目录结构

```
├── op_host
│   ├── config
│   │   ├── ascend310p
│   │   ├── ascend910_93
│   │   └── ascend910b
│   │       ├── is_inf_binary.json
│   │       └── is_inf_simplified_key.ini
│   ├── is_inf_def.cpp
│   ├── is_inf_infershape.cpp
│   ├── is_inf_tiling_arch35.cpp
│   ├── is_inf_tiling_arch35.h
│   ├── is_inf_tiling_def.h
│   ├── is_inf_tiling.cpp
│   └── CMakeLists.txt
└── op_kernel
    ├── arch35
    ├── is_inf.cpp
    └── is_inf.h
```

https://gitcode.com/cann

is_inf.cpp kernel核函数

```cpp
// kernel function
extern "C" __global__ __aicore__ void is_inf(GM_ADDR inputs, GM_ADDR outputs,
GM_ADDR workspace, GM_ADDR tiling)
{
    if (workspace == nullptr) {
        return;
    }
    GET_TILING_DATA(tilingData, tiling);

    GM_ADDR userWS = nullptr;

    const int16_t F16_INF_NUM = 0x7c00;
    const int16_t BF16_INF_NUM = 0x7f80;
    const int32_t FLOAT_INF_NUM = 0x7f800000;
    const int16_t SIGN_MASK = 0x7fff;

    if (TILING_KEY_IS(1)) {
        IsInf<half, SIGN_MASK, F16_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    } else if (TILING_KEY_IS(2)) {
        IsInf<float, SIGN_MASK, FLOAT_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    } else if (TILING_KEY_IS(3)) {
        IsInf<half, SIGN_MASK, BF16_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    }
}
```

定义不同数据类型的mask

依据不同数据类型划分tilingkey

CANN

ops-math仓算子开发-IsInf算子kernel实现

op_host及op_kernel目录结构

```
├── op_host
│   ├── config
│   │   ├── ascend310p
│   │   ├── ascend910_93
│   │   └── ascend910b
│   │       ├── is_inf_binary.json
│   │       └── is_inf_simplified_key.ini
│   ├── is_inf_def.cpp
│   ├── is_inf_infershape.cpp
│   ├── is_inf_tiling_arch35.cpp
│   ├── is_inf_tiling_arch35.h
│   ├── is_inf_tiling_def.h
│   ├── is_inf_tiling.cpp
│   └── CMakeLists.txt
└── op_kernel
    ├── arch35
    ├── is_inf.cpp
    └── is_inf.h
```

is_inf.h 算子计算逻辑

```
template <typename T, auto MASK, auto INF_MASK>
__aicore__ inline void IsInf<T, MASK, INF_MASK>::CompareInf(const int32_t
dataLength)
{
    LocalTensor<int16_t> ubX = inputQueue.DeQue<int16_t>();
    LocalTensor<uint8_t> result = outputQueue.AllocTensor<uint8_t>();
    cacheTensor = cacheTensorBuff.Get<int16_t>();

    // 和sign_mask做按位与操作
    Duplicate(cacheTensor, (int16_t)MASK, dataLength);
    And(ubX, ubX, cacheTensor, dataLength);

    uint32_t actualCalCount = dataLength / selectInterval;
    Adds(ubX, ubX, (int16_t)-INF_MASK, dataLength);
    And(ubX, ubX, cacheTensor, dataLength);
    Mins(ubX, ubX, (int16_t)1, dataLength);
    Muls(ubX, ubX, (int16_t)-1, dataLength);
    Adds(ubX, ubX, (int16_t)1, dataLength);
    Cast(result, ubX.ReinterpretCast<half>(), RoundMode::CAST_CEIL,
actualCalCount);

    inputQueue.FreeTensor(ubX);
    outputQueue.EnQue(result);
}
```


CANN

ops-math仓算子开发-IsInf算子编译安装及执行

编译： bash build.sh --pkg --soc=${soc_version} [--vendor_name=${vendor_name}] [--ops=${op_list}]

 # 以IsInf算子编译为例 # bash build.sh --pkg --soc=ascend910b --ops=is_inf

安装： ./cann-ops-math-${vendor_name}_linux-${arch}.run

执行用例： bash build.sh --run_example ${op} ${mode} ${pkg_mode} [--vendor_name=${vendor_name}]

 # 以IsInf算子example执行为例 # bash build.sh --run_example is_inf eager cust --vendor_name=custom

CANN

CANN开源开放仓库体验任务

社区任务是CANN开源开放的系列活动，本期配合“CANN算子开源周”，于9.25-9.28进行每日发放。
通过CANN开源和社区任务的结合，CANN将为开发者提供更多的分享和贡献个人成果的路径和措施。期待更多的开发者能够加入CANN生态，共同推动CANN的发展和应用。
伴随每日开源仓库，本期发布任务为“**CANN开源开放仓库体验任务**”，诚邀各位开发者前来体验。

任务说明

- **任务描述**

在CANN开源开放组织下的开源仓库下提出有效issue，包括但不限于Readme、仓库BUG等方面。开源仓库列表如下：

1. ops-math：9.25日开源
2. ops-cv：9.26日开源
3. ops-nn：9.27日开源
4. ops-trans：9.28日开源

- **获奖规则**

活动结束后根据审核后的有效issue个数，排名TOP 30发放奖品。

- **注意事项**

重复issue按最早时间计入有效issue个数。

如何参加

活动地址：https://gitcode.com/org/cann/discussions/8

参与步骤：

➢ 任务报名
在活动总页面按照模板进行评论报名，当前任务序号为1。


【任务详情】：

序号	任务值	链接	状态	承接队伍	奖品
1	100	【1】CANN开源开放仓库体验		/	CANN 单肩包

报名模板：
【队名】：根本不报错
【序号】：1
【状态】：报名

➢ issue反馈
在各个CANN开源仓库中提出issue， issue模板自行选择，标题需要增加前缀，如下所示，否则不予计入：
【CANN开源开放仓库体验】xxxx问题反馈

CANN

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态

社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem

Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.

上CANN社区获取干货

关注CANN公众号获取资讯

https://gitcode.com/cann