算子工程学习记录

lxl

2024 年 7 月 11 日

1 工程创建

CANN软件包中提供了工程创建工具msopgen，开发者可以输入算子原型定义文件生成Ascend C算子开发工程。

1.1 步骤1 编写AddCustom算子的原型定义json文件

假设AddCustom算子的原型定义文件命名为add_custom.json，文件内容如下：

```
[
{
    "op": "AddCustom",
    "input_desc": [
    {
        "name": "x",
        "param_type": "required",
        "format": ["ND"],
        "type": ["fp16"]
    },
    {
        "name": "y",
        "param_type": "required",
        "format": ["ND"],
        "type": ["fp16"]
    }
    ],
    "output_desc": [
    {
        "name": "z",
        "param_type": "required",
        "format": ["ND"],
        "type": ["fp16"]
    }
    ]
}
]
```

主要是描述算子的输入输出和名称，这些事下面msopgen创建我们自定义工程必须的信息

1.2 步骤2 使用msopgen工具生成AddCustom算子的开发工程。

如果msopgen没有被加入path的话，就得用绝对路径来定位它来运行，一般他的位置在
$HOME/Ascend/ascend-toolkit/latest/python/site-packages/bin 或者
$INSTALL_DIR/python/site-packages/bin/msopgen

1.2.1 使用方法：

```
${INSTALL_DIR}/python/site-packages/bin/msopgen gen -i $HOME/sample/add_custom.json
-c ai_core-<soc_version> -lan cpp -out $HOME/sample/AddCustom
```

- `${INSTALL_DIR}` 为 CANN 软件安装后文件存储路径，请根据实际环境进行替换。
- `-i`：算子原型定义文件 `add_custom.json` 所在路径。
- `-c: ai_core-<soc_version>` 代表算子在 AI Core 上执行，`<soc_version>` 为昇腾 AI 处理器的型号，可通过 `npu-smi info` 命令进行查询，基于同系列的 AI 处理器型号创建的算子工程，其基础功能能力通用。例如 `soc_version` 设置为 `Ascend310P1`，创建的算子工程，也可以用于开发运行于 `Ascend310P3` 上的算子。
- `-lan`：参数 `cpp` 代表算子基于 Ascend C 编程框架，使用 C++ 编程语言开发。

注意 -out后也要改成我们创建的算子名称，大驼峰命名法

1.3 步骤3 命令执行完后，会在$HOME/sample目录下生成算子工程目录AddCustom，工程中包含算子实现的模板文件，编译脚本等，如下所示：

图 1: 目录结构

2 算子主机和设备端的工作

2.1 主机侧

定义好有多少核，怎么切分数据，主要是完成把数据准备好，切分策略确定，送往设备侧运算
用到的数据结构的详细解释（待补充）

- TilingContext:
- TilingData:
- context：

2.2 设备侧

- 分给每个核的大份数据分成能被核心一次性处理的n组tile

```
__aicore__ inline void Init(GM_ADDR x, GM_ADDR y, GM_ADDR z, uint32_t
    totalLength, uint32_t tileNum)
{
    ASSERT(GetBlockNum() != 0 && "block dim can not be zero!");
    this->blockLength = totalLength / GetBlockNum();
    this->tileNum = tileNum;
    ASSERT(tileNum != 0 && "tile num can not be zero!");
    this->tileLength = this->blockLength / tileNum / BUFFER_NUM;

    xGm.SetGlobalBuffer((__gm__ DTYPE_X *)x + this->blockLength *
        GetBlockIdx(), this->blockLength);
    yGm.SetGlobalBuffer((__gm__ DTYPE_Y *)y + this->blockLength *
        GetBlockIdx(), this->blockLength);
    zGm.SetGlobalBuffer((__gm__ DTYPE_Z *)z + this->blockLength *
        GetBlockIdx(), this->blockLength);
    pipe.InitBuffer(inQueueX, BUFFER_NUM, this->tileLength * sizeof(DTYPE_X)
        );
    pipe.InitBuffer(inQueueY, BUFFER_NUM, this->tileLength * sizeof(DTYPE_Y)
        );
    pipe.InitBuffer(outQueueZ, BUFFER_NUM, this->tileLength * sizeof(DTYPE_Z
        ));
}
```

Listing 1: add_custom_init

- 处理三步骤 CopyIn，Compute，CopyOut
  CopyIn：把数据片搬入设备
  Compute：负责设备内数据运算
  CopyOut：把运算结果搬到内存
  它们看似是得等前一步执行完才能到后一步，要等很长时间，实际上是指令的发送顺序，不用等命令执行完，这些指令被很快的按这个次序发送到设备的指令队列，在那里形成流水

- 队列中buffer num的个数对性能的影响：
  - Q1: 怎么影响性能的？
  - A1：待补充，没理清楚