CANN 社区版
8.2.RC1.alpha001

Ascend Transformer 加速库接口

文档版本	01
发布日期	2025-08-04



华为技术有限公司

版权所有 © 华为技术有限公司 2025。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

 版权所有 © 华为技术有限公司

目 录

1 变更声明.....1
2 注意事项.....3
3 头文件列表.....4
4 算子调用示例.....5
5 atb/types.h.....10
5.1 公共定义.....10
5.2 Dims.....11
5.3 TensorDesc.....11
5.4 Tensor.....12
5.5 VariantPack.....12
5.6 Chunk.....13
5.7 Node.....13
5.8 GraphParam.....14
5.9 ErrorType.....15
6 atb/svector.h.....18
6.1 公共定义.....18
6.2 SVector.....18
6.2.1 SVector.....19
6.2.2 ~SVector().....19
6.2.3 at.....19
6.2.4 begin.....20
6.2.5 clear.....20
6.2.6 data.....21
6.2.7 empty.....21
6.2.8 end.....21
6.2.9 insert.....22
6.2.10 operator !=.....22
6.2.11 operator <.....23
6.2.12 operator=.....23
6.2.13 operator ==.....24
6.2.14 operator[].....24

 版权所有 © 华为技术有限公司

6.2.15 push_back 25
6.2.16 reserve 26
6.2.17 resize 26
6.2.18 size 26
7 atb/context.h 28
7.1 Context 28
7.2 CreateContext 29
7.3 DestroyContext 30
8 atb/operation.h 31
8.1 Operation 31
8.2 CreateOperation 32
8.3 DestroyOperation 33
8.4 CloneOperationParam 34
8.5 UpdateOperationParam 35
8.6 SetExecuteStreamId 36
8.7 GetExecuteStreamId 37
9 atb/operation_infra.h 38
10 atb/infer_op_params.h 40
10.1 ActivationType 42
10.2 CommMode 43
10.3 DynamicQuantType 44
10.4 InputLayout 44
10.5 QuantType 45
10.6 operator== 45
10.7 ActivationOperation 46
10.7.1 功能描述 46
10.7.2 定义 46
10.7.3 参数列表 47
10.7.4 输入输出 47
10.7.5 规格约束 50
10.7.6 功能列表 50
10.7.7 算子调用示例（C++） 57
10.8 AllGatherOperation 59
10.9 AllGatherVOperation 64
10.10 AllReduceOperation 69
10.11 AllToAllOperation 79
10.12 AllToAllVOperation 83
10.13 AllToAllVV2Operation 89
10.14 AsStridedOperation 95
10.15 BlockCopyOperation 98
10.16 BroadcastOperation 100

 版权所有 © 华为技术有限公司

10.17 CohereLayerNormOperation....102
10.18 ConcatOperation....104
10.19 CumsumOperation....107
10.20 DynamicNTKOperation....109
10.21 ElewiseOperation....111
10.21.1 功能描述....111
10.21.2 算子上下文....112
10.21.3 硬件支持情况....112
10.21.4 定义....112
10.21.5 参数列表....114
10.21.6 输入输出....115
10.21.7 规格约束....122
10.21.8 算子调用示例（C++）....123
10.22 FaUpdateOperation....125
10.23 FillOperation....126
10.24 FusedAddTopkDivOperation....129
10.25 GatherOperation....133
10.26 GatherPreRmsNormOperation....137
10.27 GatingOperation....139
10.28 GroupedMatmulInplaceAddOperation....141
10.29 GroupedMatmulWithRoutingOperation....143
10.30 GroupTopkOperation....148
10.31 IndexAddOperation....152
10.32 KVCacheOperation....156
10.33 LayerNormOperation....158
10.34 LayerNormWithStrideOperation....167
10.35 LinearOperation....169
10.35.1 功能....169
10.35.2 对标算子....170
10.35.3 算子上下文....170
10.35.4 定义....171
10.35.5 参数列表....171
10.35.6 功能列表....173
10.35.6.1 常规 matmul 与叠加偏置....173
10.35.6.2 matmul 与 add 融合....176
10.35.6.3 matmul 与 inplaceAdd 融合....179
10.35.6.4 matmul 反量化与叠加偏置....181
10.35.6.5 matmul 使能爱因斯坦乘....184
10.35.7 算子调用示例（C++）....186
10.35.7.1 linear_demo.cpp....186
10.35.7.2 linear_einsum_demo.cpp....188
10.35.7.3 linear_dequant_demo.cpp....190

 版权所有 © 华为技术有限公司

10.35.7.4 linear_dequant_inference_demo.cpp 191
10.36 LinearParallelOperation 193
10.36.1 功能描述 193
10.36.2 定义 193
10.36.3 参数列表 194
10.36.4 输入输出 196
10.36.5 规格约束 198
10.36.6 算子调用示例（C++） 199
10.36.6.1 linear_parallel_demo.cpp 199
10.36.6.2 linear_parallel_quant_demo.cpp 200
10.37 LinearSparseOperation 202
10.38 MlaPreprocessOperation 204
10.39 MultiLatentAttentionOperation 212
10.40 MultinomialOperation 215
10.41 NonzeroOperation 217
10.42 NormRopeReshapeOperation 218
10.43 OnehotOperation 220
10.44 PadOperation 222
10.45 PagedAttentionOperation 224
10.45.1 功能描述 224
10.45.2 算子上下文 224
10.45.3 计算公式 224
10.45.4 定义 224
10.45.5 参数列表 225
10.45.6 输入输出列表 227
10.45.7 规格约束 232
10.45.8 功能列表 232
10.45.8.1 各功能共存情况 232
10.45.8.2 mask 类型 233
10.45.8.3 基础功能 234
10.45.8.4 控制可计算 batch 235
10.45.8.5 多头自适应压缩（alibi 场景） 236
10.45.8.6 多头自适应压缩（rope 场景） 236
10.45.8.7 并行解码 237
10.45.8.8 MLA 238
10.45.8.9 反量化融合 238
10.45.8.10 qkv 全量化 238
10.45.8.11 logN 缩放 239
10.45.8.12 BNSD 维度输入 240
10.45.8.13 MLA 合并输入 kvcache 功能 241
10.45.8.14 MLA 并行解码 241
10.45.9 算子调用示例（C++） 243

 版权所有 © 华为技术有限公司

10.45.9.1 paged_attention_demo.cpp 243
10.45.9.2 paged_attetion_inference_demo.cpp 245
10.46 RecvOperation 247
10.47 ReduceOperation 249
10.48 ReduceScatterOperation 252
10.49 ReduceScatterVOperation 256
10.50 RelayAttentionOperation 259
10.51 RepeatOperation 262
10.52 ReshapeAndCacheOperation 264
10.52.1 功能 264
10.52.2 算子上下文 265
10.52.3 定义 265
10.52.4 参数列表 266
10.52.5 输入输出 266
10.52.5.1 不开启多头压缩功能 266
10.52.5.2 开启 alibi 场景下的多头压缩功能 267
10.52.5.3 开启 rope 场景下的多头压缩功能 269
10.52.5.4 开启单入单出功能 270
10.52.5.5 开启支持输入转 NZ 格式功能 271
10.52.6 规格约束 272
10.52.7 算子调用示例（C++） 273
10.52.7.1 reshape_and_cache_demo.cpp 273
10.52.7.2 reshape_and_cache_inference_demo.cpp 276
10.53 ReshapeAndCacheOmniOperation 278
10.54 ReshapeAndCacheWithStrideOperation 281
10.54.1 功能 281
10.54.2 算子上下文 282
10.54.3 定义 282
10.54.4 参数列表 283
10.54.5 输入输出 283
10.54.5.1 不开启多头压缩功能 283
10.54.5.2 开启单入单出功能 285
10.54.6 规格约束 285
10.55 RmsNormOperation 286
10.55.1 功能描述 286
10.55.2 使用场景 286
10.55.3 计算公式 287
10.55.4 定义 288
10.55.5 参数列表 289
10.55.6 输入输出 292
10.55.7 规格约束 296
10.55.8 功能列表 297

 版权所有 © 华为技术有限公司

10.55.9 算子调用示例（C++）..... 297
10.56 RmsNormWithStrideOperation..... 299
10.57 RopeOperation..... 305
10.58 RopeQConcatOperation..... 310
10.59 SelfAttentionOperation..... 313
10.59.1 功能描述..... 313
10.59.2 算子上下文..... 313
10.59.3 计算图..... 314
10.59.4 计算公式..... 316
10.59.5 定义..... 316
10.59.6 参数列表..... 317
10.59.7 输入输出列表..... 321
10.59.8 规格约束..... 328
10.59.9 功能列表..... 329
10.59.9.1 各功能共存情况..... 329
10.59.9.2 控制可计算 batch..... 330
10.59.9.3 全量/增量分离..... 331
10.59.9.4 高精度..... 331
10.59.9.5 clamp 缩放..... 331
10.59.9.6 压缩 mask..... 332
10.59.9.7 kv-bypass..... 335
10.59.9.8 logN 缩放..... 336
10.59.9.9 qkv 全量化..... 336
10.59.9.10 BNSD 维度输入..... 338
10.59.9.11 kv tensorlist 格式输入..... 341
10.59.9.12 MLA 合并输入 kvCache 功能..... 341
10.59.9.13 Sliding window attention（SWA）功能..... 342
10.59.9.14 FA 支持 Prefix Encoder..... 346
10.59.10 算子调用示例（C++）..... 346
10.59.10.1 fa_encoder_demo.cpp..... 346
10.59.10.2 fa_encoder_inference_demo.cpp..... 349
10.59.10.3 fa_pa_encoder_demo.cpp..... 351
10.59.10.4 fa_prefix_encoder_demo.cpp..... 353
10.60 SendOperation..... 356
10.61 SetValueOperation..... 357
10.62 SliceOperation..... 360
10.63 SoftmaxOperation..... 363
10.64 SortOperation..... 365
10.65 SplitOperation..... 367
10.66 SwigluQuantOperation..... 371
10.67 TopkToppSamplingOperation..... 373
10.67.1 功能..... 373

 版权所有 © 华为技术有限公司

10.67.2 定义 373
10.67.3 参数列表 374
10.67.4 输入输出 375
10.67.5 规格约束 378
10.67.6 功能列表 378
10.67.6.1 非 batch 级随机种子、topk 取样 378
10.67.6.2 batch 级别随机种子、topk 的 multinomial 取样 379
10.67.6.3 batch 级随机种子，topk 的 Exponential 取样 380
10.67.6.4 batch 级别随机种子、topk 的 logprobs multinomial 取样 380
10.67.6.5 batch 级随机种子，topk 的 logprobs Exponential 取样 383
10.68 TransdataOperation 385
10.69 TransposeOperation 390
10.70 UnpadOperation 393
10.71 WhereOperation 395
11 atb/train_op_params.h 398
11.1 operator== 398
11.2 FastSoftMaxOperation 399
11.3 FastSoftMaxGradOperation 403
11.4 GenAttentionMaskOperation 406
11.5 LaserAttentionOperation 407
11.6 LaserAttentionGradOperation 410
11.7 PadWithHiddenStateOperation 413
11.8 RmsNormBackwardOperation 414
11.9 RopeGradOperation 417
11.10 StridedBatchMatmulOperation 419
11.11 UnpadWithHiddenStateOperation 421
12 atb/common_op_params.h 423
12.1 operator== 423
12.2 EventOperation 423
13 atb/graph_op_builder.h 426
13.1 GraphOpBuilder 426
13.1.1 GraphOpBuilder 426
13.1.2 ~GraphOpBuilder 426
13.1.3 Init 427
13.1.4 Reshape 427
13.1.5 AddOperation 428
13.1.6 Build 428
13.2 CreateGraphOpBuilder 429
13.3 DestroyGraphOpBuilder 429
14 atb/utils.h 430
14.1 Utils 430

 版权所有 © 华为技术有限公司

15 atb/comm.h 431
15.1 CreateHcclComm 431
15.2 CreateHcclCommByRankTableFile 432
15.3 DestoryHcclComm 432
15.4 CreateHcclCrossMulitComm 433
16 ATB Python API 使用说明 435
16.1 OpParam 435
16.1.1 说明 435
16.1.2 AclDataType 435
16.1.3 AclFormat 436
16.1.4 QuantType 436
16.1.5 DynamicQuantType 437
16.1.6 ActivationType 437
16.1.7 InputLayout 437
16.1.8 ActivationParam 438
16.1.9 ElewiseParam 438
16.1.10 GatherParam 440
16.1.11 LayerNormParam 441
16.1.12 LinearParam 443
16.1.13 RopeParam 444
16.1.14 SelfAttentionParam 445
16.1.15 RmsNormParam 448
16.1.16 SoftmaxParam 450
16.1.17 SplitParam 451
16.2 Operation 451
17 术语&缩略语 454

 版权所有 © 华为技术有限公司

1 变更声明

本章节的接口变更说明包括新增、修改、删除和退出声明。接口变更只体现代码层面的修改，不包含文档本身在语言、格式、链接等方面的优化改进。

- 新增：表示此次版本新增的接口。
- 修改：表示本接口相比于上个版本有修改。
- 删除：表示该接口在此次版本已经删除。
- 退出声明：表示该接口自作出退出声明的版本起停止演进，且在后续版本退出并删除。

新增特性

类名/API原型	变更说明
AllGatherVOperation	新增OP。
AllToAllVV2Operation	新增OP。
CohereLayerNormOperation	新增OP。
FaUpdateOperation	新增OP。
FusedAddTopkDivOperation	新增OP。
GatherPreRmsNormOperation	新增OP。
LayerNormWithStrideOperation	新增OP。
MlaPreprocessOperation	新增OP。
MultiLatentAttentionOperation	新增OP。
NormRopeReshapeOperation	新增OP。
RelayAttentionOperation	新增OP。
ReduceScatterVOperation	新增OP。
ReshapeAndCacheOmniOperation	新增OP。

 版权所有 © 华为技术有限公司

类名/API原型	变更说明
ReshapeAndCacheWithStrideOperation	新增OP。
RmsNormWithStrideOperation	新增OP。
RopeQConcatOperation	新增OP。
SwigluQuantOperation	新增OP。
atb/operation.h • SetExecuteStreamId • GetExecuteStreamId atb/operation_infra.h	新增图内多流并行特性。
atb/comm.h	支持创建和删除通信域。
atb/common_op_params.h	新增图间同步特性。
ATB Python API使用说明	新增python使用说明。

修改特性

类名/API原型	变更说明
atb/infer_op_params.h/ QuantType	枚举项QUANT_UNDEFINED变更为QUANT_UNQUANT，QUANT_UNDEFINED将在后续版本删除。
AllReduceOperation	其内部枚举类QuantType中枚举项QUANT_TYPE_UNDEFINED变更为QUANT_TYPE_UNQUANT，QUANT_TYPE_UNDEFINED将在后续版本删除。
LinearParallelOperation	其内部枚举类QuantType中枚举项QUANT_TYPE_UNDEFINED变更为QUANT_TYPE_UNQUANT，QUANT_TYPE_UNDEFINED将在后续版本删除。
PagedAttentionOperation	其内部枚举类QuantType中枚举项TYPE_QUANT_UNDEFINED变更为TYPE_QUANT_UNQUANT，TYPE_QUANT_UNDEFINED将在后续版本删除。
SelfAttentionOperation	其内部枚举类QuantType中枚举项TYPE_QUANT_UNDEFINED变更为TYPE_QUANT_UNQUANT，TYPE_QUANT_UNDEFINED将在后续版本删除。

 版权所有 © 华为技术有限公司

2 注意事项

版本配套

使用MindIE的推理引擎调用ATB加速库的场景：

- CANN版本≥8.0.0且MindIE版本≥1.0.0时，直接使用MindIE的推理引擎调用ATB加速库。在需要回退CANN版本时，回退后的CANN版本不能<8.0.0。
- CANN版本＜8.0.0或MindIE版本＜1.0.0时，在需要升级或者回退CANN版本时，请基于目标CANN版本重新编译调用ATB的程序，才能正常使用ATB加速库。

支持的产品型号

Atlas 200I/500 A2 推理产品

Atlas 推理系列产品

Atlas 训练系列产品

Atlas 800I A2 推理产品/Atlas A2 训练系列产品

说明

算子默认支持的产品型号为Atlas 推理系列产品、Atlas 800I A2 推理产品/Atlas A2 训练系列产品，实际以算子规格说明为准。

算子说明

- Atlas 推理系列产品 中不支持ACL_BF16（bf16）类型数据。

 版权所有 © 华为技术有限公司

3 头文件列表

头文件目录	说明
atb/types.h	基本数据结构和常量定义。
atb/svector.h	栈上的Vector定义。
atb/context.h	Context接口类，主要用于设置Operation上下文，如stream，tiling属性等。
atb/operation.h	算子类接口定义。
atb/infer_op_params.h	推理算子参数定义。
atb/train_op_params.h	训练算子参数定义。
atb/graph_op_builder.h	定义图算子的构建方法。
atb/utils.h	工具接口定义。
atb/atb_infer.h	仅包含对上述头文件的引用，引用该头文件即可提供对上述所有文件的引用。
atb/atb_comm.h	加速库通信域相关接口。

版权所有 © 华为技术有限公司

4 算子调用示例

用例整体介绍

本用例为ATB单算子示例。

该用例整体包括：

- 准备atb::VariantPack的输入tensor部分。
- 准备算子参数并创建算子。
- 主函数：通用设置卡号、创建context、设置stream部分；算子Setup，执行；清理内存，释放资源。

目录结构介绍

```
atb-linear-demo 示例工程
|-- demo_util.h
|-- linear_demo.cpp
```

Linear 算子调用示例

- **demo_util.h**

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#define CHECK_STATUS(status)                                                    \
    do {                                                                        \
        if ((status) != 0) {                                                    \
            std::cout << __FILE__ << ":" << __LINE__ << " [error]: " << (status) << std::endl; \
            exit(1);                                                            \
        }                                                                       \
    } while (0)

#define CHECK_STATUS_EXPR(status, expr)                                         \
    do {                                                                        \
        if ((status) != 0) {                                                    \
            std::cout << __FILE__ << ":" << __LINE__ << " [error]: " << (status) << std::endl; \
            expr;                                                               \
        }                                                                       \
    } while (0)

/**
```

 版权所有 © 华为技术有限公司

```
 * @brief 创建一个Tensor对象
 * @param  dataType 数据类型
 * @param  format 数据格式
 * @param  shape 数据shape
 * @return atb::Tensor 返回创建的Tensor对象
 */
atb::Tensor CreateTensor(const aclDataType dataType, const aclFormat format, std::vector<int64_t>
shape)
{
    atb::Tensor tensor;
    tensor.desc.dtype = dataType;
    tensor.desc.format = format;
    tensor.desc.shape.dimNum = shape.size();
    // tensor的dim依次设置为shape中元素
    for (size_t i = 0; i < shape.size(); i++) {
        tensor.desc.shape.dims[i] = shape.at(i);
    }
    tensor.dataSize = atb::Utils::GetTensorSize(tensor);  // 计算Tensor的数据大小
    CHECK_STATUS(aclrtMalloc(&tensor.deviceData, tensor.dataSize,
aclrtMemMallocPolicy::ACL_MEM_MALLOC_HUGE_FIRST));
    return tensor;
}

/**
 * @brief 进行数据类型转换，调用Elewise的cast Op
 * @param contextPtr context指针
 * @param stream stream
 * @param inTensor 输入tensor
 * @param outTensorType 输出tensor的数据类型
 * @param shape 输出tensor的shape
 * @return atb::Tensor 转换后的tensor
 */
atb::Tensor CastOp(atb::Context *contextPtr, aclrtStream stream, const atb::Tensor inTensor,
    const aclDataType outTensorType, std::vector<int64_t> shape)
{
    uint64_t workspaceSize = 0;
    void *workspace = nullptr;
    // 创建Elewise的ELEWISE_CAST
    atb::infer::ElewiseParam castParam;
    castParam.elewiseType = atb::infer::ElewiseParam::ELEWISE_CAST;
    castParam.outTensorType = outTensorType;
    atb::Operation *castOp = nullptr;
    CHECK_STATUS(CreateOperation(castParam, &castOp));
    atb::Tensor outTensor = CreateTensor(outTensorType, aclFormat::ACL_FORMAT_ND, shape);  // cast
输出tensor
    atb::VariantPack castVariantPack;                                                    // 参数包
    castVariantPack.inTensors = {inTensor};
    castVariantPack.outTensors = {outTensor};
    // 在Setup接口调用时对输入tensor和输出tensor进行校验。
    CHECK_STATUS(castOp->Setup(castVariantPack, workspaceSize, contextPtr));
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc(&workspace, workspaceSize,
aclrtMemMallocPolicy::ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // ELEWISE_CAST执行
    CHECK_STATUS(castOp->Execute(castVariantPack, (uint8_t *)workspace, workspaceSize,
contextPtr));
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspace));  // 清理工作空间
    }
    return outTensor;
}

/**
 * @brief 简单封装，拷贝vector data中数据以创建tensor
 * @details 用于创建outTensorType类型的tensor
 * @param contextPtr context指针
 * @param stream stream
```

 版权所有 © 华为技术有限公司

```
 * @param data 输入vector数据
 * @param outTensorType 期望输出tensor数据类型
 * @param format 输出tensor的格式，即NZ，ND等
 * @param shape 输出tensor的shape
 * @return atb::Tensor 返回创建的tensor
 */
template <typename T>
atb::Tensor CreateTensorFromVector(atb::Context *contextPtr, aclrtStream stream, std::vector<T> data,
    const aclDataType outTensorType, const aclFormat format, std::vector<int64_t> shape,
    const aclDataType inTensorType = ACL_DT_UNDEFINED)
{
    atb::Tensor tensor;
    aclDataType intermediateType;
    switch (outTensorType) {
        case aclDataType::ACL_FLOAT:
        case aclDataType::ACL_FLOAT16:
        case aclDataType::ACL_BF16:
            intermediateType = aclDataType::ACL_FLOAT;
            break;
        default:
            intermediateType = outTensorType;
    }
    if (inTensorType == outTensorType && inTensorType != ACL_DT_UNDEFINED) {
        intermediateType = outTensorType;
    }
    tensor = CreateTensor(intermediateType, format, shape);
    CHECK_STATUS(aclrtMemcpy(
        tensor.deviceData, tensor.dataSize, data.data(), sizeof(T) * data.size(),
ACL_MEMCPY_HOST_TO_DEVICE));
    if (intermediateType == outTensorType) {
        // 原始创建的tensor类型，不需要转换
        return tensor;
    }
    return CastOp(contextPtr, stream, tensor, outTensorType, shape);
}
```

- **linear_demo.cpp**

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include <atb/atb_infer.h>

#include "demo_util.h"
const int32_t DEVICE_ID = 0;
const uint32_t X_DIM_0 = 2;
const uint32_t X_DIM_1 = 3;
const uint32_t WEIGHT_DIM_0 = 3;
const uint32_t WEIGHT_DIM_1 = 2;
const uint32_t BIAS_DIM_0 = 2;

/**
 * @brief 准备atb::VariantPack
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建shape为[2, 3]的输入x tensor
    atb::Tensor xFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>{1, 2, 3, 4, 5, 6},
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {X_DIM_0, X_DIM_1});
    // 创建shape为[3, 2]的输入weight tensor
```

 版权所有 © 华为技术有限公司

```
    atb::Tensor weightFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>{1, 2, 3, 4, 5, 6},
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {WEIGHT_DIM_0, WEIGHT_DIM_1});
    // 创建shape为[2]的输入bias tensor
    atb::Tensor biasFloat = CreateTensorFromVector(
        contextPtr, stream, std::vector<float>(BIAS_DIM_0, 1.0), ACL_FLOAT16,
aclFormat::ACL_FORMAT_ND, {1, BIAS_DIM_0});
    atb::SVector<atb::Tensor> inTensors = {xFloat, weightFloat, biasFloat};
    return inTensors;
}

/**
 * @brief 创建一个linear operation
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *CreateLinearOperation()
{
    atb::infer::LinearParam param;
    param.transposeA = false;
    param.transposeB = false;
    param.hasBias = true;
    param.outDataType = aclDataType::ACL_DT_UNDEFINED;
    param.enAccum = false;
    param.matmulType = atb::infer::LinearParam::MATMUL_UNDEFINED;
    atb::Operation *LinearOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &LinearOp));
    return LinearOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    atb::Context *context = nullptr;
    void *stream = nullptr;

    CHECK_STATUS(aclInit(nullptr));
    CHECK_STATUS(aclrtSetDevice(DEVICE_ID));
    CHECK_STATUS(atb::CreateContext(&context));
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 创建op
    atb::Operation *linearOp = CreateLinearOperation();
    // 准备输入tensor
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    // 准备输出tensor
    atb::Tensor output = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {X_DIM_0,
WEIGHT_DIM_1});
    variantPack.outTensors = {output};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(linearOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // linear执行
    linearOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放资源
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
```

 版权所有 © 华为技术有限公司

```
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(linearOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    std::cout << "Linear demo success!" << std::endl;
    return 0;
}
```

用例运行

步骤1 调用前需设置ascend-toolkit、atb环境变量：

```
source ${toolkit安装目录}/set_env.sh # 如source /usr/local/Ascend/ascend-toolkit/set_env.sh
source ${nnal安装目录}/atb/set_env.sh # 如source /usr/local/Ascend/nnal/atb/set_env.sh
```

步骤2 按以下步骤编译：

```
g++ -I "${ATB_HOME_PATH}/include" -I "${ASCEND_HOME_PATH}/include" -L "${ATB_HOME_PATH}/lib" -L "${ASCEND_HOME_PATH}/lib64" xxxx.cpp demo_util.h -l atb -l ascendcl -o xxxx
```

步骤3 执行用例：

```
./xxxx
```

----结束

说明

- 使用hccl算子时，需要额外增加编译选项：-D_GLIBCXX_USE_CXX11_ABI=0。
- 使用反向等训练算子时，需要动态链接："${ATB_HOME_PATH}/lib/libatb_train.so"，即更新编译命令为：
  ```
  g++ ... demo_util.h "${ATB_HOME_PATH}/lib/libatb_train.so" ...
  ```

 版权所有 © 华为技术有限公司

5 atb/types.h

公共定义
Dims
TensorDesc
Tensor
VariantPack
Chunk
Node
GraphParam
ErrorType

5.1 公共定义

定义

```
using Status = int32_t;
using ReshapeFunc = std::function< void(const Dims &oldShape, Dims &newShape)>
using InferShapeFunc = std::function< Status(const SVector< TensorDesc > &inTensorDescs, SVector<
TensorDesc > &outTensorDescs)>
constexpr uint32_t MAX_DIM = 8;
```

参数列表

常量名称	描述
Status	类型定义，用一系列状态值表示加速库中的返回值。
ReshapeFunc	类型定义，Reshape功能，改变Tensor的shape。
InferShapeFunc	类型定义，根据输入tensor信息推导输出tensor信息。
MAX_DIM	变量，数据最大维度定义。

版权所有 © 华为技术有限公司

5.2 Dims

功能

Shape维度信息。

定义

```
struct Dims {
    int64_t dims[MAX_DIM];
    uint64_t dimNum = 0;
};
```

参数列表

成员名称	数据类型	默认值	描述
dims	int64_t	-	每一维的大小，要求大于0。
dimNum	uint64_t	0	Tensor的维数，取值范围为(0, 8]。此默认值不可用，用户需配置此项参数。

5.3 TensorDesc

功能

包含对Tensor的相关描述信息：每个Tensor的数据类型，数据排布格式和形状维度信息。

定义

```
struct TensorDesc {
    aclDataType dtype = ACL_DT_UNDEFINED;
    aclFormat format = ACL_FORMAT_UNDEFINED;
    Dims shape;
};
```

参数列表

成员名称	数据类型	默认值	描述
dtype	aclDataType	ACL_DT_UNDEFINED	Tensor数据类型。 具体请参见aclDataType。 Atlas 推理系列产品 中不支持ACL_BF16（bf16）类型数据。
format	aclFormat	ACL_FORMAT_UNDEFINED	Tensor数据排布格式。 具体请参见aclFormat。

 版权所有 © 华为技术有限公司

成员名称	数据类型	默认值	描述
shape	Dims	-	Tensor数据的形状，Tensor的维度信息须满足Dims的参数约束要求，具体请参见5.2 Dims。

5.4 Tensor

功能

加速库的Tensor定义，包含每个Tensor的描述符、NPU内存地址、CPU内存地址和内存大小等。

定义

```
struct Tensor {
    TensorDesc desc;
    void *deviceData = nullptr;
    void *hostData = nullptr;
    uint64_t dataSize = 0;
};
```

参数列表

成员名称	数据类型	默认值	描述
desc	TensorDesc	-	Tensor描述信息。
deviceData	void *	nullptr	TensorNPU内存地址。
hostData	void *	nullptr	TensorCPU内存地址。
dataSize	uint64_t	0	“deviceData”或“hostData”指向内容的内存大小。

5.5 VariantPack

功能

加速库算子执行时需要构造VariantPack存放输入及最终输出。

定义

```
struct VariantPack {
    SVector<Tensor> inTensors;
    SVector<Tensor> outTensors;
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	数据类型	默认值	描述
inTensors	SVector<Tensor>	-	存放所有输入tensor的SVector。
outTensors	SVector<Tensor>	-	存放所有输出tensor的SVector。

5.6 Chunk

功能

在Host侧对Tensor执行Split切分操作。

定义

```
struct Chunk{
    uint32_t chunkNum = 1;
    uint32_t chunkIndex = 0;
}
```

参数列表

成员名称	数据类型	默认值	说明
chunkNum	uint32_t	1	切分数量。
chunkIndex	uint32_t	0	标记均分后使用的是第几份数据。

5.7 Node

功能

图算子中的Operation节点，每个Node表示一个Operation或者GraphOperation，所有的Node组成一个完整的图算子。

定义

```
struct Node {
    Operation* operation = nullptr;
    SVector<uint32_t> inTensorIds;
    SVector<uint32_t> outTensorIds;
    SVector<ReshapeFunc> inTensorReshapeFuncs;
    SVector<Chunk> inTensorChunks;
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	数据类型	默认值	描述
operation	Operation	nullptr	Node对应的operation或者graphOperation。
inTensorIds	SVector<uint32_t >	-	Node对应的operation或者graphOperation的输入tensorId SVector。
outTensorIds	SVector<uint32_t >	-	Node对应的operation或者graphOperation的输出tensorId SVector。
inTensorReshapeFuncs	SVector<ReshapeFunc >	-	Node对应的operation或者graphOperation的每个输入tensor的reshape函数SVector。
inTensorChunks	SVector<Chunk >	-	存放chunk。

说明

inTensorIds、outTensorIds和inTensorReshapeFuncs均为SVector，每个元素的顺序需要和对应Tensor的顺序保持一致。

5.8 GraphParam

功能

图算子参数。

定义

```
struct GraphParam {
    std::string name;
    uint32_t inTensorNum = 0;
    uint32_t outTensorNum = 0;
    uint32_t internalTensorNum = 0;
    std::vector<Node> nodes;
    InferShapeFunc inferShapeFunc = nullptr;
};
```

参数列表

成员名称	数据类型	默认值	描述
name	std::string	-	图算子名称。 仅允许字母、数字、下划线，名称长度不超过128。

 版权所有 © 华为技术有限公司

成员名称	数据类型	默认值	描述
inTensorNum	uint32_t	0	图算子输入tensor的数量，需小于或等于256，传入不正确的Tensor数量将返回“ERROR_INVALID_IN_TENSOR_NUM”报错，具体请参见**5.9 ErrorType**。
outTensorNum	uint32_t	0	图算子输出tensor的数量，需小于或等于256，传入不正确的Tensor数量将返回“ERROR_INVALID_IN_TENSOR_NUM”报错，具体请参见**5.9 ErrorType**。
internalTensorNum	uint32_t	0	图算子中间tensor的数量，需小于或等于256，传入不正确的Tensor数量将返回“ERROR_INVALID_IN_TENSOR_NUM”报错，具体请参见**5.9 ErrorType**。
nodes	std::vector<Node>	-	图算子Node Vector。 nodes的长度满足小于1024。 nodes的顺序，需要满足各个node对应operation的执行顺序依赖关系，先执行的在前面，后执行的在后面。 如果inTensorNum，outTensorNum与internalTensorNum之和为S，则nodes中各个元素的inTensorIds，outTensorIds中各元素值均要求落在[0, S-1]范围内。
inferShapeFunc	InferShapeFunc	nullptr	inferShape函数指针。

5.9 ErrorType

功能

加速库日志中定义的错误类型。

定义

```
enum ErrorType : int {
    NO_ERROR = 0,
    ERROR_INVALID_PARAM,
    ERROR_INVALID_GRAPH,
    ERROR_INTERNAL_ERROR,
    ERROR_RT_FAIL,
    ERROR_INVALID_IN_TENSOR_NUM,
    ERROR_INVALID_TENSOR_DTYPE,
    ERROR_INVALID_TENSOR_FORMAT,
    ERROR_INVALID_TENSOR_DIM,
    ERROR_INVALID_TENSOR_SIZE,
    ERROR_OPERATION_NULL_RUNNER,
    ERROR_GRAPH_INFERSHAPE_FUNC_FAIL,
    ERROR_CANN_ERROR,
    ERROR_INVALID_TENSOR_INI_MATCH,
    ERROR_INVALID_TENSOR_ADDR,
    ERROR_INVALID_TENSOR_NUM,
```

 版权所有 © 华为技术有限公司

```
    ERROR_INVALID_TENSOR_DIM_NUM,
    ERROR_INVALID_SINGLE_OPERATION_PARAM,
    ERROR_GRAPH_NODE_RESHAPE_FUNC_FAIL,
    ERROR_INVALID_GRAPH_NODE_CHUNK,
    ERROR_INVALID_CONTEXT_ADDR,
    ERROR_INVALID_STREAM,
    ERROR_INVALID_WORKSPACE_SIZE,
    ERROR_INVALID_WORKSPACE_ADDR,
    ERROR_INVALID_OPERATION_ADDR,
    ERROR_HCCL_FAIL,
    ERROR_OUT_OF_DEVICE_MEMORY,
    ERROR_OUT_OF_HOST_MEMORY,
    ERROR_COMM_EMPTY,
    ERROR_COPY_HOST_MEMORY_FAIL
};
```

参数列表

成员名称	描述
NO_ERROR	正确。
ERROR_INVALID_PARAM	无效参数。
ERROR_INVALID_GRAPH	计算图错误。
ERROR_INTERNAL_ERROR	内部错误。
ERROR_RT_FAIL	调用Runtime接口失败。
ERROR_INVALID_IN_TENSOR_NUM	算子输入tensor数量与定义不一致。
ERROR_INVALID_TENSOR_DTYPE	tensor数据类型错误。
ERROR_INVALID_TENSOR_FORMAT	tensor数据格式错误。
ERROR_INVALID_TENSOR_DIM	tensor数据维度错误。
ERROR_INVALID_TENSOR_SIZE	tensor数据size错误。
ERROR_OPERATION_NULL_RUNNER	Operation内部错误。
ERROR_GRAPH_INFERSHAPE_FUNC_FAIL	图infershapeFunc错误。
ERROR_CANN_ERROR	调用CANN接口错误。
ERROR_INVALID_TENSOR_INI_MATCH	算子输入或参数检验失败。
ERROR_INVALID_TENSOR_ADDR	tensor地址错误。
ERROR_INVALID_TENSOR_NUM	tensor数量错误。

 版权所有 © 华为技术有限公司

成员名称	描述
ERROR_INVALID_TENSOR_DIM_NUM	tensor维度数错误。
ERROR_INVALID_SINGLE_OPERATION_PARAM	单Operation参数错误。
ERROR_GRAPH_NODE_RESHAPE_FUNC_FAIL	图节点reshapeFuncs设置有误。
ERROR_INVALID_GRAPH_NODE_CHUNK	图节点Chunk参数错误。
ERROR_INVALID_CONTEXT_ADDR	HOST侧算子运行上下文地址有误， 通常是分配失败导致的。
ERROR_INVALID_STREAM	一般是流ID错误，可能是没有分配流ID或使用了错误的流ID。
ERROR_INVALID_WORKSPACE_SIZE	Workspace大小有误。
ERROR_INVALID_WORKSPACE_ADDR	Workspace地址有误，通常是未分配地址。
ERROR_INVALID_OPERATION_ADDR	Operation地址有误。
ERROR_HCCL_FAIL	HCCL接口调用失败。
ERROR_OUT_OF_DEVICE_MEMORY	Device内存不足。
ERROR_OUT_OF_HOST_MEMORY	Host内存不足。
ERROR_COMM_EMPTY	通信域指针为空。
ERROR_COPY_HOST_MEMORY_FAIL	Host内存拷贝失败。

 版权所有 © 华为技术有限公司

6 atb/svector.h

公共定义

SVector

6.1 公共定义

定义

```
constexpr size_t MAX_SVECTOR_SIZE = 256;
constexpr bool CHECK_BOUND = true;
struct MaxSizeExceeded : public std::exception {};
constexpr size_t atb::DEFAULT_SVECTOR_SIZE = 64;
std::ostream &operator << (std::ostream &os, const SVector< T > &svector )
```

参数列表

成员名称	描述
MAX_SVECTOR_SIZE	最大容器大小。
CHECK_BOUND	默认为“true”，即默认进行边界检验。
MaxSizeExceeded	越界异常参数，具体情况请参见各接口描述。
DEFAULT_SVECTOR_SIZE	默认容器大小。
operator<<()	函数，输出容器中的元素。

6.2 SVector

 版权所有 © 华为技术有限公司

6.2.1 SVector

功能

SVector构造函数。

定义

```
constexpr SVector()
SVector(std::initializer_list<T> list)
SVector(std::size_t size, const T &value = 0)
SVector(const SVector<T> &other)
```

参数列表

成员名称	输入/输出	说明
list	输入	以初始化列表来初始化SVector。
size	输入	SVector初始化时大小。
value	输入	SVector初始化时数值。默认为“0”。
other	输入	使用已有SVector值给当前SVector赋值。

规格约束

若不使用默认构造，要求传入的“list”长度或者“size”的大小需小于**DEFAULT_SVECTOR_SIZE**，否则会抛出长度溢出异常。

6.2.2 ~SVector()

功能

析构函数。

定义

```
~SVector()
```

返回值说明

无。

6.2.3 at

功能

访问指定位置的元素。

定义

```
T &at(std::size_t i)
const T &at(std::size_t i) const
```

 版权所有 © 华为技术有限公司

参数列表

参数名	输入/输出	说明
i	输入	需要返回的是第“i”位元素。

返回值说明

类型	输入/输出	说明
T&	输出	返回的元素。

规格约束

若调用函数时，SVector中元素数量为0或者传入的“i”大于等于SVector中元素数量，会抛出越界异常。

6.2.4 begin

功能

获取SVector中起始元素地址。

定义

```
T *begin() noexcept
const T *begin() const noexcept
```

返回值说明

类型	输入/输出	说明
T *	输出	SVector首部指针。

6.2.5 clear

功能

清空SVector中所有元素。

定义

```
void clear() noexcept
```

 版权所有 © 华为技术有限公司

6.2.6 data

功能

获取SVector起始元素地址。

定义

```
T *data() noexcept
const T *data() const noexcept
```

返回值说明

类型	输入/输出	说明
T *	输出	Svector的数据地址。

6.2.7 empty

功能

判断SVector是否为空。

定义

```
bool empty() const noexcept
```

返回值说明

类型	输入/输出	说明
bool	输出	• 返回“true”时SVector为**空**。 • 返回“false”则SVector**非空**。

6.2.8 end

功能

获取SVector尾元素地址。

定义

```
T *end() noexcept
const T *end() const noexcept
```

 版权所有 © 华为技术有限公司

返回值说明

类型	输入/输出	说明
T *	输出	SVector尾部指针。

6.2.9 insert

功能

向SVector内指定位置插入元素。

定义

```
void insert(const std::size_t pos, const T &value) noexcept((!CHECK_BOUND) &&
std::is_nothrow_assignable<T, const T &>::value)
```

参数列表

参数名	输入/输出	说明
pos	输入	在pos处插入元素。
value	输入	插入在pos处的元素值。

规格约束

若调用函数时，待插入元素的SVector长度大于或等于**MAX_SVECTOR_SIZE**，则会抛出长度溢出异常。

若传入的参数“pos”大于SVector内元素的数量，则会抛出长度溢出异常。

6.2.10 operator !=

功能

判断两个Svector的元素是否存在不同。

定义

```
bool operator != (const SVector<T> &other) const
```

参数列表

参数名	输入/输出	说明
other	输入	判断两个Svector中的元素是否存在不同。

 版权所有 © 华为技术有限公司

返回值说明

类型	输入/输出	说明
bool	输出	• 返回“true”时**不相同**。 • 返回“false”时**相同**。

6.2.11 operator <

功能

判断一个SVector中的元素是否比另一个SVector小。

定义

```
bool operator < (const SVector<T> &other) const
```

参数列表

参数名	输入/输出	说明
other	输入	判断一个容器中的元素是否比另一个容器小。

返回值说明

类型	输入/输出	说明
bool	输出	• 返回“true”时**小于**。 • 返回“false”时**不小于**。

6.2.12 operator=

功能

重载运算符函数，将初始化列表中的元素赋值给一个SVector对象。

定义

```
SVector &operator=(std::initializer_list<T> list)
SVector& &operator=(const SVector &other)
```

 版权所有 © 华为技术有限公司

参数列表

参数名	输入/输出	说明
list	输入	将初始化列表中的元素赋值给一个SVector。
other	输入	用一个容器给另一个容器赋值。

返回值说明

类型	输入/输出	说明
SVector&	输出	返回容器引用。

6.2.13 operator ==

功能

判断两个SVector中的元素是否全部相同。

定义

```
bool operator == (const SVector<T> &other) const
```

参数列表

参数名	输入/输出	说明
other	输入	判断成员与“other”是否相同。

返回值说明

类型	输入/输出	说明
bool	输出	• 返回“true”时**相同**。 • 返回“false”时**不相同**。

6.2.14 operator[]

功能

访问指定位置的元素。

 版权所有 © 华为技术有限公司

定义

```
T &operator[](std::size_t i)
const T &operator[](std::size_t i) const
```

参数列表

参数名	输入/输出	说明
i	输入	需要返回的是第“i”位元素。

返回值说明

类型	输入/输出	说明
T&	输出	返回的元素。

规格约束

调用函数时，SVector中元素数量为0或者传入的“i”大于等于SVector中元素数量，会抛出越界异常。

6.2.15 push_back

功能

插入元素到指定SVector。

定义

```
void push_back(const T &val) noexcept((!CHECK_BOUND) && std::is_nothrow_assignable<T, const T &>::value)
```

参数列表

参数名	输入/输出	说明
val	输入	添加在SVector内元素的值。

规格约束

调用函数时，待添加SVector内元素必须小于**MAX_SVECTOR_SIZE**，否则接口将会抛出长度溢出异常。

 版权所有 © 华为技术有限公司

6.2.16 reserve

功能

改变SVector容量大小，将SVector内部数据清空，并将SVector容量大小定义为size大小。

定义

```
void reserve (std::size_t size)
```

参数列表

参数名	输入/输出	说明
size	输入	用于预分配内存空间。

规格约束

SVector默认容量为**DEFAULT_SVECTOR_SIZE**，传入的size需大于**DEFAULT_SVECTOR_SIZE**且小于**MAX_SVECTOR_SIZE**，反之，则会抛出异常。

6.2.17 resize

功能

改变SVector容器大小，不能改变SVector容量。

定义

```
void resize(std::size_t size)
```

参数列表

参数名	输入/输出	说明
size	输入	将SVector容器大小改为“size”。

规格约束

若调用函数时，传入的“size”参数不能超过SVector容量，反之，则会抛出异常。

6.2.18 size

功能

获取SVector的大小。

 版权所有 © 华为技术有限公司

定义

```
std::size_t size() const noexcept
```

返回值说明

类型	输入/输出	说明
size	输出	获取容器的大小。

 版权所有 © 华为技术有限公司

7 atb/context.h

Context

CreateContext

DestroyContext

7.1 Context

功能

加速库上下文类，主要用于管理Operation运行所需要的全局资源。

Context类会管理任务流队列比如Operation执行以及TilingCopy，管理tiling内存的申请与释放。

定义

```
class Context {
public:
    Context() = default;
    virtual ~Context() = default;
    virtual Status SetExecuteStream(aclrtStream stream) = 0;
    virtual aclrtStream GetExecuteStream() const = 0;
    virtual Status SetAsyncTilingCopyStatus(bool enable) = 0;
    virtual bool GetAsyncTilingCopyStatus() const = 0;
    virtual Status SetExecuteStreams(const std::vector<aclrtStream> &streams) = 0;
    virtual std::vector<aclrtStream> GetExecuteStreams() = 0;
    virtual Status SetExecuteType(ExecuteType type) = 0;
    virtual ExecuteType GetExecuteType() = 0;
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	参数	返回值	描述
SetExecuteStream	stream	Status	stream表示了Device侧的执行流， 每个流具体表现为一条执行队列。 通过这个函数设置使用传入的流执行对应的Operation。 ● stream：传入的stream队列。 如果设置成功，返回NO_ERROR。如果失败，请参见**5.9 ErrorType**。
GetExecuteStream	-	aclrtStream	获取当前执行stream。
SetAsyncTilingCopyStatus	enable	Status	设置异步拷贝tiling信息功能是否开启，如果是，则创建Stream和Event来进行tiling拷贝过程。 ● enable：传入的标志，bool类型。 如果设置成功，返回NO_ERROR。如果失败，请参见**5.9 ErrorType**。
GetAsyncTilingCopyStatus	-	bool	获取tiling拷贝状态，如果获取成功，返回True。
SetExecuteStreams	streams	Status	设置实际的执行流，Operation执行时会根据配置的streamId从Context匹配对应的实际执行流。 ● streams：需要设置的一组stream。 状态值，如果设置成功，返回NO_ERROR。如果失败，请参见**5.9 ErrorType**。
GetExecuteStreams	-	std::vector<aclrtStream>	获取Context中当前设置的一组执行流。 如果设置成功，返回Context当前设置的一组执行流。
SetExecuteType	-	Status	设置Execute的类型。
GetExecuteType	-	ExecuteType	获取当前context Execute的类型。

7.2 CreateContext

功能

创建上下文。

在当前进程或线程中显式创建一个Context。

 版权所有 © 华为技术有限公司

定义

```
Status CreateContext(Context **context);
```

参数列表

参数名	输入/输出	说明
context	输入	传入指向context的二级指针， 函数内部会将其指向新创建的context。

返回值说明

类型	输入/输出	说明
Status	输出	状态值。如果设置成功，返回NO_ERROR。如果失败，请参见5.9 ErrorType。

7.3 DestroyContext

功能

销毁上下文中所有的资源。

定义

```
Status DestroyContext(Context *context);
```

参数列表

参数名	输入/输出	说明
context	输入	指向context的一级指针。

返回值说明

类型	输入/输出	说明
Status	输出	状态值。如果设置成功，返回NO_ERROR。如果失败，请参见5.9 ErrorType。

 版权所有 © 华为技术有限公司

8 atb/operation.h

Operation

CreateOperation

DestroyOperation

CloneOperationParam

UpdateOperationParam

SetExecuteStreamId

GetExecuteStreamId

8.1 Operation

功能

加速库Operation类。

该接口类定义了算子准备与执行需要的一系列的接口，用户通过创建Operation对象来执行算子。

定义

```
class Operation {
public:
    Operation() = default;
    virtual ~Operation() = default;
    virtual std::string GetName() const = 0;
    virtual Status InferShape(const SVector<TensorDesc> &inTensorDescs, SVector<TensorDesc>
&outTensorDescs) const = 0;
    virtual uint32_t GetInputNum() const = 0;
    virtual uint32_t GetOutputNum() const = 0;
    virtual Status Setup(const VariantPack &variantPack, uint64_t &workspaceSize, Context *context) = 0;
    virtual Status Execute(const VariantPack &variantPack, uint8_t *workspace, uint64_t workspaceSize,
Context *context) = 0;
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	描述
GetName	获取创建的Operation的名字。
InferShape	根据输入tensor描述信息推导出输出tensor的描述信息。 ● “inTensorDescs”存放所有输入tensor描述信息的SVector。 ● “outTensorDescs”存放所有输出tensor描述信息的SVector。 如果执行成功，返回NO_ERROR。其他错误码表示失败，具体详见**5.9 ErrorType**。
GetInputNum	获取Op/GraphOp输入tensor个数接口。
GetOutputNum	获取Op/GraphOp输出tensor个数接口。
Setup	Operation执行前的一系列准备工作，主要是计算Operation执行过程需要分配的内存空间workspaceSize。 ● variantPack：输入与输出tensor。 ● workspaceSize：获取Operation执行需要分配的内存空间。 ● context：Operation执行准备工作所在的上下文。 如果执行成功，返回NO_ERROR。其他错误码表示失败，具体详见**5.9 ErrorType**。
Execute	Operation执行的流程，根据Setup过程中得到的“workspaceSize”为Operation执行分配实际的内存，并执行Operation。 ● variantPack：输入与输出tensor。 ● workspace：Operation执行分配的内存地址。 ● workspaceSize：Operation执行需要分配的内存空间。 ● context：Operation执行所在的上下文。 如果执行成功，返回NO_ERROR。其他错误码表示失败，具体详见**5.9 ErrorType**。

8.2 CreateOperation

功能

创建Operation。

定义

```
template <typename OpParam> Status CreateOperation(const OpParam &opParam, Operation **operation);
```

 版权所有 © 华为技术有限公司

参数列表

参数名称	输入/输出	描述
opParam	输入	创建Operation的参数信息，具体参数类型可以引用10 atb/infer_op_params.h和11 atb/train_op_params.h文件。
operation	输出	指向新创建Operation的二级指针。

返回值说明

类型	输入/输出	说明
Status	输出	返回NO_ERROR表示成功，返回其它值表示失败。

规格约束

与8.3 DestroyOperation接口配对使用，先调用CreateOperation接口创建再调用DestroyOperation接口进行销毁。

调用CreateOperation创建的Operation在使用完毕后，需调用8.3 DestroyOperation接口销毁，否则将导致内存泄漏。

支持10 atb/infer_op_params.h和11 atb/train_op_params.h的参数。

8.3 DestroyOperation

功能

销毁Operation。

定义

```
Status DestroyOperation(Operation *operation);
```

参数列表

参数名称	输入/输出	描述
operation	输入	Operation的指针。

 版权所有 © 华为技术有限公司

返回值说明

类型	输入/输出	说明
Status	输出	返回NO_ERROR表示成功，返回其它值表示失败。

规格约束

与8.2 CreateOperation接口配对使用，先调用CreateOperation接口再调用DestroyOperation接口。

调用8.2 CreateOperation创建的Operation在使用完毕后，需调用DestroyOperation接口销毁，否则将导致内存泄漏。

支持10 atb/infer_op_params.h和11 atb/train_op_params.h的参数。

8.4 CloneOperationParam

功能

拷贝Operation的Param参数到第二个入参。

定义

```
Status CloneOperationParam(const Operation *operation, OpParam &opParam);
```

参数列表

参数名称	描述
operation	Operation指针。
opParam	OpParam的引用，将返回operation的opParam浅拷贝。

返回值说明

类型	输入/输出	说明
Status	输出	状态值，如果成功，返回NO_ERROR。

规格约束

支持11 atb/train_op_params.h的参数和12.2 EventOperation。

 版权所有 © 华为技术有限公司

使用示例

步骤1 创建Operation。

```
train::FastSoftMaxParam param;
param.headNum = 3;
param.qSeqLen = { 10, 20, 30 };
atb::Operation *op = nullptr;
atb::Status st = atb::CreateOperation(param, &op);
```

步骤2 获取当前Operation的param。

```
train::FastSoftMaxParam cloneParam;
st = atb::CloneOperationParam(op, cloneParam);
```

----结束

8.5 UpdateOperationParam

功能

更新Operation的Param参数。

定义

```
Status UpdateOperationParam(Operation *operation, const OpParam &opParam);
```

参数列表

参数名称	描述
operation	Operation指针。
opParam	Operation需要更新的新param值。

返回值说明

类型	输入/输出	说明
Status	输出	状态值，如果成功，返回NO_ERROR。

规格约束

支持**11 atb/train_op_params.h**的参数和**12.2 EventOperation**。

使用示例

步骤1 创建Operation。

```
train::FastSoftMaxParam param;
param.headNum = 3;
param.qSeqLen = { 10, 20, 30 };
atb::Operation *op = nullptr;
atb::Status st = atb::CreateOperation(param, &op);
```

 版权所有 © 华为技术有限公司

步骤2 获取当前Operation的param。

```
train::FastSoftMaxParam cloneParam;
st = atb::CloneOperationParam(op, cloneParam);
```

步骤3 更新当前Operation的param。

```
cloneParam.headNum = 66;
st = atb::UpdateOperationParam(op, cloneParam);
```

----结束

8.6 SetExecuteStreamId

功能

设置Operation使用的streamId，这里的streamId为调用Context的SetExecuteStreams接口时传入streams中的序号。（例如如果需要op1在stream2上运行（streams={stream1，stream2，stream3}），那么这里调用为SetExecuteStreamId(op1, 1)）。

定义

```
Status SetExecuteStreamId(Operation *operation, uint32_t streamId);
```

参数列表

参数名称	输入/输出	描述
operation	输入	被设置的Operation指针。
streamId	输入	需要设置的streamId。

返回值说明

类型	输入/输出	说明
Status	输出	状态值，返回NO_ERROR表示成功，返回其它值表示失败。

规格约束

若自定义插件算子（PluginOperation）需要使用该功能，PluginOperation需要继承自OperationInfra接口基类。

说明

该接口对单算子与图算子的设置行为略有差异，单算子调用该接口时修改单算子本身的StreamId，图算子调用该接口会修改图中所有streamId为0的Operation。默认的streamId为0。

 版权所有 © 华为技术有限公司

8.7 GetExecuteStreamId

功能

获取Operation使用的streamId。

定义

```
uint32_t GetExecuteStreamId(Operation *operation);
```

参数列表

参数名称	输入/输出	描述
operation	输入	需要获取streamId的Operation指针。

返回值说明

类型	输入/输出	说明
uint32_t	输出	该Operation当前使用的streamId。

规格约束

若PluginOperation需要使用该功能，PluginOperation需要继承自OperationInfra接口基类。

说明

图算子调用该接口获取的streamId无法代表整体的streamId，因为图内的各个Operation有可能不在同一条stream上。默认的streamId为0。

 版权所有 © 华为技术有限公司

9 atb/operation_infra.h

功能

定义ATB的OperationInfra类。该接口类定义了加速库插件算子的基类，基类提供了一些内置公共接口供用户编写自己的插件算子。

定义

```
class OperationInfra : public Operation {
public:
    OperationInfra();
    OperationInfra(const OperationInfra &other);
    OperationInfra& operator = (const OperationInfra &other);
    ~OperationInfra() override;
    void SetExecuteStreamId(uint32_t streamId);
    uint32_t GetExecuteStreamId() const;
    aclrtStream GetExecuteStream(Context *context);
private:
    std::unique_ptr<OperationImpl> impl_;
};
```

参数列表

成员名称	参数	返回值	描述
OperationInfra	-	-	构造函数。
OperationInfra	other	-	拷贝构造函数。 ● other：拷贝的对象。
operator =	other	OperationInfra &	赋值构造函数。 ● other：赋值的对象。 如果执行成功，返回对象本身。
~OperationInfra	-	-	析构函数。
SetExecuteStreamId	streamId	-	设置该Operation需要使用的stream的id，id为context中的stream序号，该接口需要配合GetExecuteStream接口使用。 ● streamId：需要设置的stream的id。

 版权所有 © 华为技术有限公司

成员名称	参数	返回值	描述
GetExecuteStreamId	-	uint32_t	获取当前Operation使用的stream的id。 如果执行成功，返回stream的id，id为context中的stream序号。
GetExecuteStream	context	aclrtStream	获取当前Operation使用的stream。 ● context：Operation使用的Context。 如果执行成功，返回当前Operation使用的stream。

 版权所有 © 华为技术有限公司

10 atb/infer_op_params.h

ActivationType
CommMode
DynamicQuantType
InputLayout
QuantType
operator==
ActivationOperation
AllGatherOperation
AllGatherVOperation
AllReduceOperation
AllToAllOperation
AllToAllVOperation
AllToAllVV2Operation
AsStridedOperation
BlockCopyOperation
BroadcastOperation
CohereLayerNormOperation
ConcatOperation
CumsumOperation
DynamicNTKOperation
ElewiseOperation
FaUpdateOperation
FillOperation

 版权所有 © 华为技术有限公司

FusedAddTopkDivOperation

GatherOperation

GatherPreRmsNormOperation

GatingOperation

GroupedMatmulInplaceAddOperation

GroupedMatmulWithRoutingOperation

GroupTopkOperation

IndexAddOperation

KVCacheOperation

LayerNormOperation

LayerNormWithStrideOperation

LinearOperation

LinearParallelOperation

LinearSparseOperation

MlaPreprocessOperation

MultiLatentAttentionOperation

MultinomialOperation

NonzeroOperation

NormRopeReshapeOperation

OnehotOperation

PadOperation

PagedAttentionOperation

RecvOperation

ReduceOperation

ReduceScatterOperation

ReduceScatterVOperation

RelayAttentionOperation

RepeatOperation

ReshapeAndCacheOperation

ReshapeAndCacheOmniOperation

ReshapeAndCacheWithStrideOperation

RmsNormOperation

RmsNormWithStrideOperation

 版权所有 © 华为技术有限公司

RopeOperation
RopeQConcatOperation
SelfAttentionOperation
SendOperation
SetValueOperation
SliceOperation
SoftmaxOperation
SortOperation
SplitOperation
SwigluQuantOperation
TopkToppSamplingOperation
TransdataOperation
TransposeOperation
UnpadOperation
WhereOperation

10.1 ActivationType

说明

激活函数类型枚举。

定义

```
enum ActivationType : int {
    ACTIVATION_UNDEFINED = 0,
    ACTIVATION_RELU,
    ACTIVATION_GELU,
    ACTIVATION_FAST_GELU,
    ACTIVATION_SWISH,
    ACTIVATION_LOG,
    ACTIVATION_SWIGLU_FORWARD,
    ACTIVATION_SWIGLU_BACKWARD,
    ACTIVATION_SIGMOID,
    ACTIVATION_FASTER_GELU_FORWARD,
    ACTIVATION_MAX,
};
```

成员

成员名称	描述
ACTIVATION_UNDEFINED	未定义。
ACTIVATION_RELU	RELU激活类型。

 版权所有 © 华为技术有限公司

成员名称	描述
ACTIVATION_GELU	GELU激活类型。
ACTIVATION_FAST_GELU	FAST_GELU激活类型。 快速运算的Gelu激活函数，对Tensor内每个element做Gelu激活函数近似计算，计算速度更快，同时保持较高的准确性。
ACTIVATION_SWISH	SWISH激活类型。
ACTIVATION_LOG	LOG激活类型。
ACTIVATION_SWIGLU_FORWARD	SWIGLU_FORWARD激活类型。 Swiglu正向激活函数。对于Atlas 推理系列产品，仅支持32位对齐的数据。
ACTIVATION_SWIGLU_BACKWARD	SWIGLU_BACKWARD激活类型。 Swiglu正向激活函数的反向，求梯度时使用。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
ACTIVATION_SIGMOID	SIGMOID激活类型。
ACTIVATION_FASTER_GELU_FORWARD	FASTER_GELU_FORWARD激活类型。
ACTIVATION_MAX	枚举最大值, 非激活类型。

10.2 CommMode

说明

通信算子支持的通信模式。

定义

```
enum CommMode : int {
    COMM_UNDEFINED = -1,
    COMM_MULTI_PROCESS,
    COMM_MULTI_THREAD,
};
```

成员

成员名称	描述
COMM_UNDEFINED	未定义。
COMM_MULTI_PROCESS	多进程通信。
COMM_MULTI_THREAD	多线程通信。

 版权所有 © 华为技术有限公司

10.3 DynamicQuantType

说明

动态量化支持的类型。

定义

```
enum atb::infer::DynamicQuantType:int{
    DYNAMIC_QUANT_UNDEFINED = 0,
    DYNAMIC_QUANT_SYMMETRIC,
    DYNAMIC_QUANT_ASYMMETRIC
};
```

成员

成员名称	描述
DYNAMIC_QUANT_UNDEFINED	非动态量化。
DYNAMIC_QUANT_SYMMETRIC	对称动态量化。
DYNAMIC_QUANT_ASYMMETRIC	非对称动态量化，暂不支持。

10.4 InputLayout

说明

数据排布类型。

定义

```
enum InputLayout : int {
    TYPE_BSND = 0,
    TYPE_BNSD
};
```

成员

成员名称	描述
TYPE_BSND	默认值，表示数据排布为BSND。
TYPE_BNSD	表示数据排布为BNSD。

 版权所有 © 华为技术有限公司

10.5 QuantType

说明

量化支持的类型。

定义

```
enum QuantType: int {
    QUANT_UNDEFINED = 0,
    QUANT_UNQUANT = 0,
    QUANT_INT4,
    QUANT_INT8,
    QUANT_INT16,
    QUANT_FLOAT8,
    QUANT_FLOAT16,
};
```

成员

成员名称	描述
QUANT_UNDEFINED	不量化。
QUANT_UNQUANT	不量化。
QUANT_INT4	INT4量化，当前不支持。
QUANT_INT8	INT8量化。
QUANT_INT16	INT16量化，当前不支持。
QUANT_FLOAT8	FLOAT8量化，当前不支持。
QUANT_FLOAT16	FLOAT16量化，当前不支持

10.6 operator==

功能

判断两个RopeParam是否相同。

定义

```
inline bool operator == (const RopeParam &left, const RopeParam &right)
{
   return left.rotaryCoeff == right.rotaryCoeff && left.cosFormat == right.cosFormat;
}
inline bool operator==(const AllGatherVParam &left, const AllGatherVParam &right)
{
    return left.rank == right.rank && left.rankSize == right.rankSize && left.rankRoot == right.rankRoot &&
           left.hcclComm == right.hcclComm && left.commMode == right.commMode && left.backend ==
right.backend &&
           left.rankTableFile == right.rankTableFile && left.commDomain == right.commDomain;
}
```

 版权所有 © 华为技术有限公司

```
inline bool operator==(const ReduceScatterVParam &left, const ReduceScatterVParam &right)
{
    return left.rank == right.rank && left.rankSize == right.rankSize && left.rankRoot == right.rankRoot &&
        left.sendCounts == right.sendCounts && left.sdispls == right.sdispls && left.recvCount ==
right.recvCount &&
        left.reduceType == right.reduceType && left.hcclComm == right.hcclComm && left.commMode ==
right.commMode &&
        left.backend == right.backend && left.rankTableFile == right.rankTableFile &&
        left.commDomain == right.commDomain;
}
inline bool operator==(const RopeQConcatParam &left, const RopeQConcatParam &right)
{
    (void)left;
    (void)right;
    return true;
}
inline bool operator==(const AllToAllVV2Param &left, const AllToAllVV2Param &right)
{
    return left.rank == right.rank && left.rankSize == right.rankSize && left.rankRoot == right.rankRoot &&
        left.hcclComm == right.hcclComm && left.commMode == right.commMode && left.backend ==
right.backend &&
        left.rankTableFile == right.rankTableFile && left.commDomain == right.commDomain;
}
```

10.7 ActivationOperation

10.7.1 功能描述

激活函数，可以引入非线性因素。

图 10-1 ActivationOperation



10.7.2 定义

```
struct ActivationParam {
    enum GeLUMode : int{
        TANH_MODE = 0,
        NONE_MODE,
    };
    ActivationType activationType=ACTIVATION_UNDEFINED;
    float scale = 1.0f; // for Swish
    int32_t dim = -1;   // for Swiglu
    GeLUMode geluMode = TANH_MODE;
    uint8_t rsv[8] = {0};
};
```

 版权所有 © 华为技术有限公司

10.7.3 参数列表

成员名称	类型	默认值	描述
activationType	ActivationType	ACTIVATION_UNDEFINED	激活类型。 ● ACTIVATION_UNDEFINED：未定义，此默认类型不可用，用户需配置此项参数。 ● ACTIVATION_RELU：RELU激活类型。 ● ACTIVATION_GELU：GELU激活类型。 ● ACTIVATION_FAST_GELU：快速运算的Gelu激活类型。 ● ACTIVATION_SWISH：SWISH激活类型。 ● ACTIVATION_LOG：LOG激活类型。 ● ACTIVATION_SWIGLU_FORWARD：Swiglu正向激活函数。 ● ACTIVATION_SWIGLU_BACKWARD：Swiglu_forward反向激活函数，求梯度时使用。 ● ACTIVATION_SIGMOID：Sigmoid激活类型。 ● ACTIVATION_FASTER_GELU_FORWARD：FASTER_GELU_FORWARD激活类型。 ● ACTIVATION_MAX：枚举最大值, 非激活类型。
scale	float	1.0f	Swish激活函数中的缩放系数。
dim	int32_t	-1	Swiglu正反向激活函数使用, 延最后一维切分 ，即只支持传入-1。
geluMode	GeLUMode	TANH_MODE	Gelu模型选择。 ● TANH_MODE: Tanh近似计算。 ● NONE_MODE: 非近似计算。
rsv[8]	uint8_t	{0}	预留参数。

10.7.4 输入输出

RELU 激活类型

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float/bf16	ND	输入

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
output	[dim_0, dim_1, ..., dim_n]	float/bf16	ND	输出 和输入维度/类型/格式相同。

FAST_GELU 激活类型

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输入
output	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输出 和输入维度/类型/格式相同。

LOG 激活类型

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND	输入
output	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND	输出 和输入维度/类型/格式相同。

SIGMOID 激活类型

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输入
output	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输出 和输入维度/类型/格式相同。
Atlas 200I/500 A2 推理产品上仅支持float16数据类型。				

 版权所有 © 华为技术有限公司

SWISH 激活类型

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输入
output	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输出 和输入维度/类型/格式相同。

GELU 激活类型

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND/NZ	输入
output	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND/NZ	输出 和输入维度/类型/格式相同。

SWIGLU_FORWARD 激活类型

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND/NZ	输入 当“activationType”为ACTIVATION_SWIGLU_FORWARD时：对于Atlas 推理系列产品，最后一维需为32的倍数。
output	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND/NZ	输出 和输入维度/类型/格式相同，其dim维所对应的大小是输入的一半。

 版权所有 © 华为技术有限公司

SWIGLU_BACKWARD 激活类型

参数	维度	数据类型	格式	描述
x1	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND	当“activationType”为ACTIVATION_SWIGLU_BACKWARD时，不支持Atlas 推理系列产品。
x2	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND	输入 和x1的维度/类型/格式相同，其dim维所对应的大小是输入x1的2倍。
output	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND	输出 和x1的维度/类型/格式相同，其dim维所对应的大小是输入x1的2倍。

FASTER_GELU_FORWARD 激活类型

参数	维度	数据类型	格式	描述
input	[dim_0, dim_1, ..., dim_n]	float/float16/bf16	ND / NZ	输入tensor。
output	[dim_0, dim_1, ..., dim_n]	float/float16/bf16	ND / NZ	输出tensor。维度、数据类型、数据格式与输入tensor一致。

10.7.5 规格约束

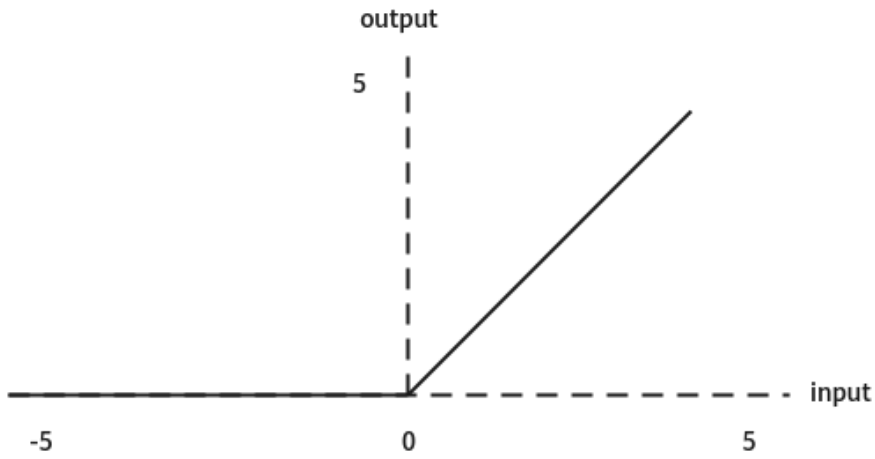
激活函数类型请参考**10.1 ActivationType**。

10.7.6 功能列表

RELU 激活类型功能

- **使用参数**
  activationType = ACTIVATION_RELU
- **函数图**

 版权所有 © 华为技术有限公司



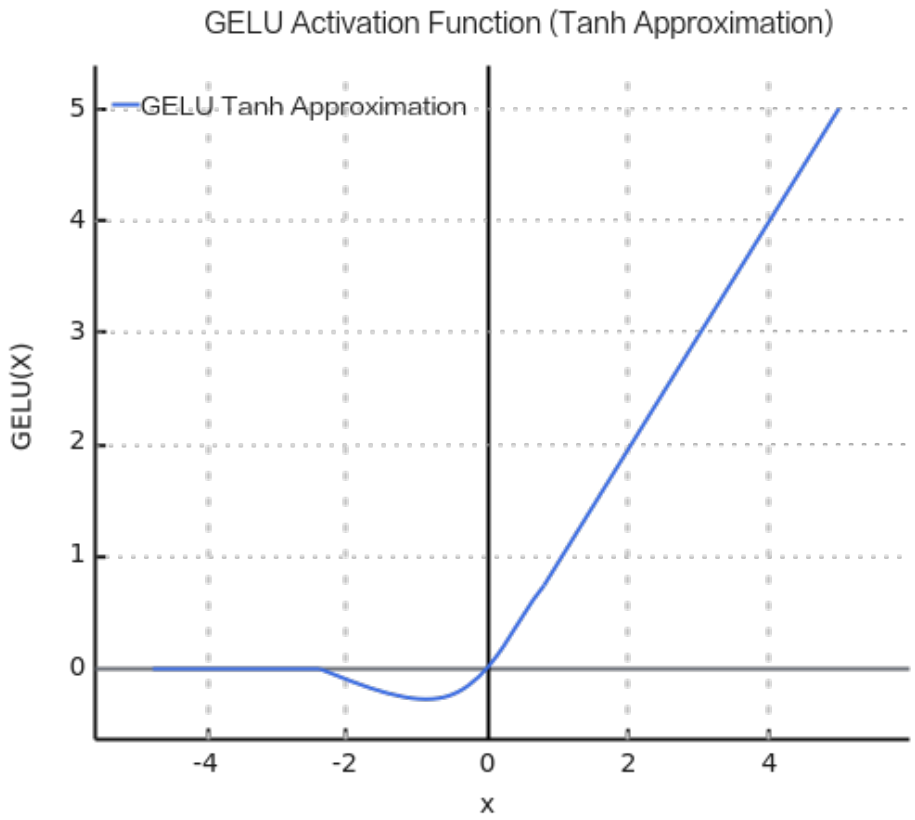
- **表达式**

$$\sigma\left(x\right)=\begin{cases}max(0,x), & x>=0\\0, & x<0\end{cases}$$

GELU 激活类型功能

- **Tanh近似计算**
  - **使用参数**
    - activationType = ACTIVATION_GELU
    - geluMode = TANH_MODE
  - **函数图**

 版权所有 © 华为技术有限公司

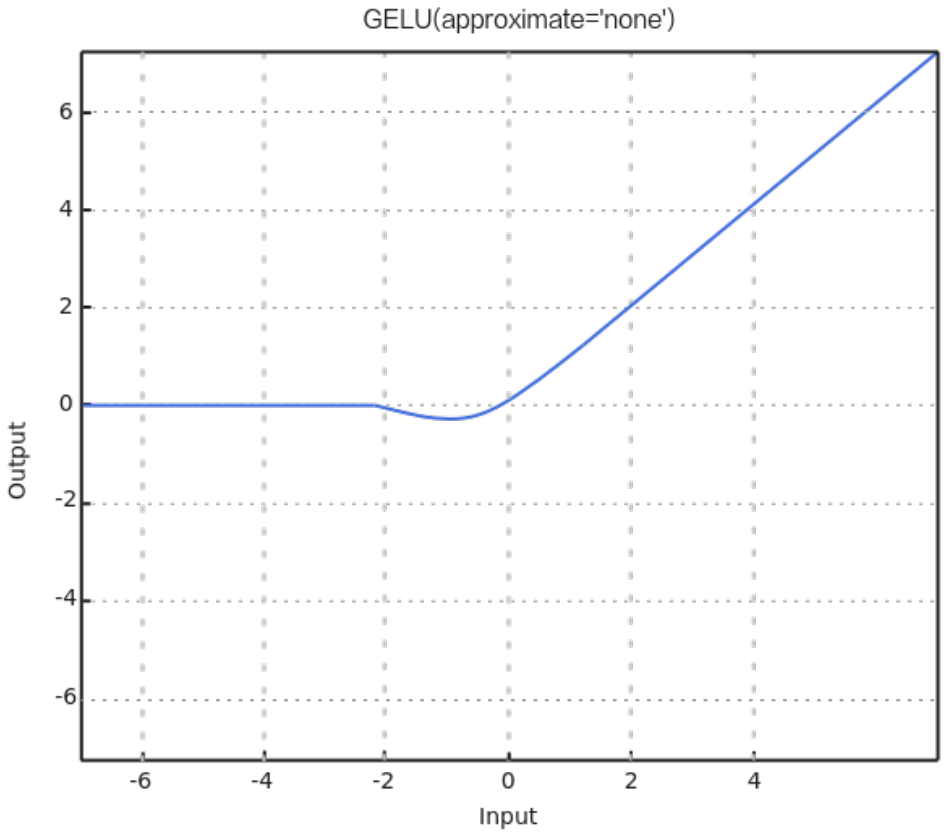


  - **表达式**

$$GELU(x) = 0.5 * x * (1 + Tanh(\sqrt{2/\pi} * (x + 0.044715 * x^3)))$$

- **非近似计算**
  - **使用参数**
    - activationType = ACTIVATION_GELU
    - geluMode = NONE_MODE
  - **函数图**

 版权所有 © 华为技术有限公司



- **表达式**

$$GELU(x) = x * \Phi(x)$$

$\Phi(x)$ 是高斯分布的累积分布函数

$$\Phi\left(x \right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt$$

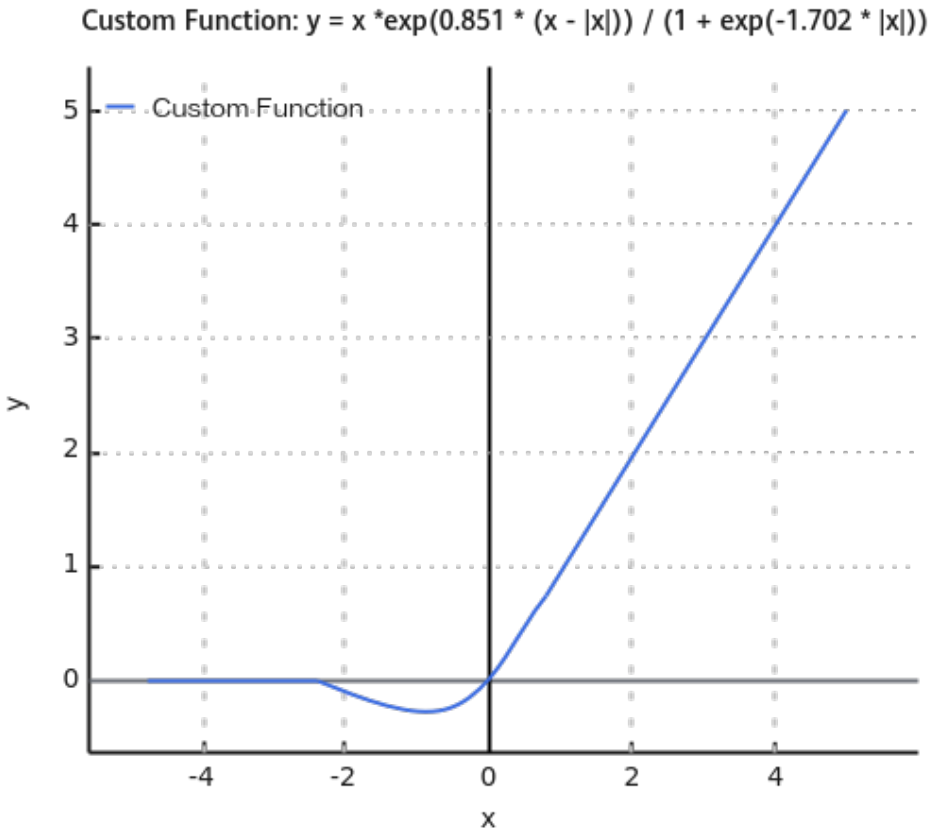
FAST_GELU 激活类型功能

- **使用参数**

  activationType = ACTIVATION_FAST_GELU

- **函数图**

 版权所有 © 华为技术有限公司



- **表达式**

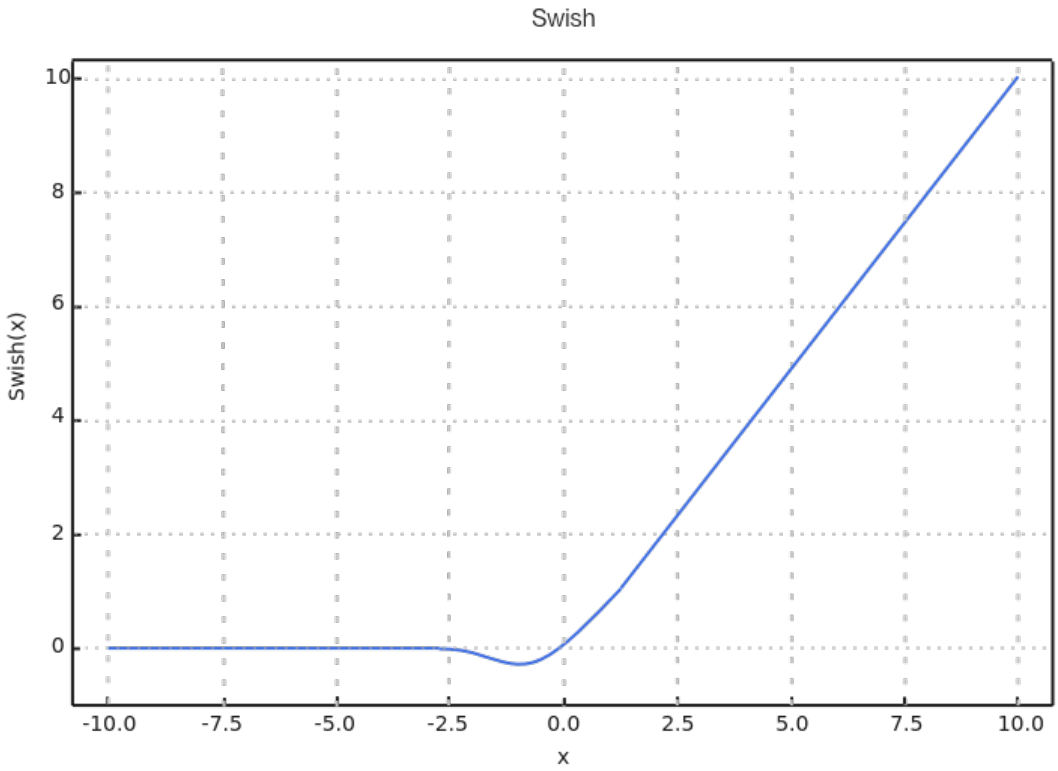
$$y = \frac{x * e^{0.851*(x-|x|)}}{1 + e^{-1.702*|x|}}$$

SWISH 激活类型功能

- **使用参数**
  - activationType = ACTIVATION_SWISH
  - scale
- **函数图**

  跟所输入的参数scale有关，当scale=1时

 版权所有 © 华为技术有限公司



- **表达式**

$$y = \frac{x}{1 + exp^{-x * scale}}$$

LOG 激活类型功能

- **使用参数**

  activationType = ACTIVATION_LOG

- **表达式**

$$y_i = log(x_i)$$

SWIGLU_FORWARD 激活类型功能

对于Atlas 推理系列产品，仅支持32位对齐的数据。

- **使用参数**
  - activationType = ACTIVATION_SWIGLU_FORWARD
  - dim
- **表达式**

  a, b为对输入tensor延最后一维进行均分。

 版权所有 © 华为技术有限公司

$$y = \frac{b * a}{1 + exp^{-a}}$$

- **规格约束**
  dim只可以延最后一维进行均分，必须传入-1。

SWIGLU_BACKWARD 激活类型功能

当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

- **使用参数**
  - activationType = ACTIVATION_SWIGLU_BACKWARD
  - dim
- **表达式**
  x1为输入tensor1
  x2为输入tensor2
  a，b为对x2延最后一维进行均分

$$y_1 = b * x_1 \frac{1 + (a + 1) * exp^{-a}}{(1 + exp^{-a})^2}$$

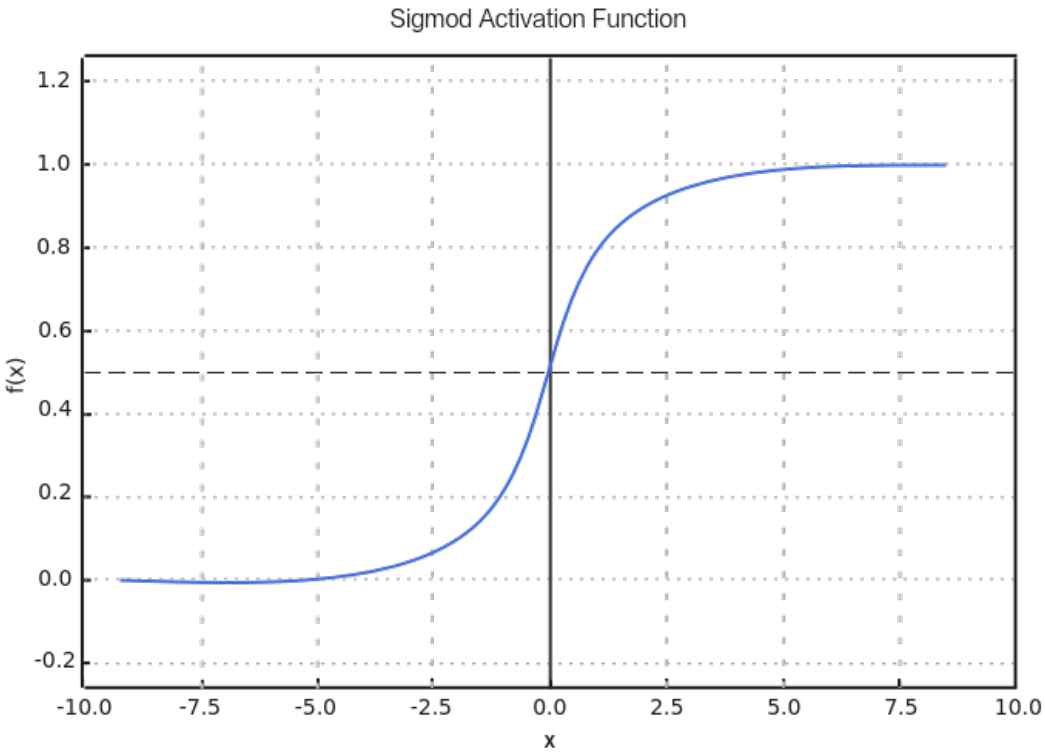
$$y_2 = \frac{x_1 * a}{1 + exp^{-a}}$$

- **规格约束**
  dim只可以延最后一维进行均分，必须传入-1。

SIDMOD 激活类型功能

- **使用参数**
  activationType = ACTIVATION_SIGMOID
- **函数图**

 版权所有 © 华为技术有限公司



- **表达式**

$$y = \frac{1}{1 + exp^{-x}}$$

FASTER_GELU_FORWARD 激活类型功能

- **使用参数**

  activationType = ACTIVATION_FASTER_GELU_FORWARD

- **表达式**

$$y = x * SIGMOID(1.702 * |x|) * exp^{0.851 * (x - |x|)}$$

10.7.7 算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include <random>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"
#include "demo_util.h"
const uint32_t BATCH_SIZE = 16;     // 批处理大小
const uint32_t SEQ_LEN = 1024;      // 序列长度
```

```
const uint32_t HIDDEN_SIZE = 4096;  // 隐藏层维度
/**
 * @brief 准备atb::VariantPack中的输入tensor
 * @return atb::SVector<atb::Tensor> 返回{[BATCH_SIZE, SEQ_LEN, HIDDEN_SIZE]}的输入tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor()
{
    // 创建一个[BATCH_SIZE*SEQ_LEN*HIDDEN_SIZE]的vector，其中各个值为取值范围为[-100,100)的随机数
    std::vector<float> inTensorData(BATCH_SIZE * SEQ_LEN * HIDDEN_SIZE);
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<float> dis(-100.0f, 100.0f);
    for (float &val : inTensorData) {
        val = dis(gen);
    }
    // 创建输入tensor
    atb::Tensor inTensor = CreateTensor(ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE, SEQ_LEN,
HIDDEN_SIZE});
    CHECK_STATUS(aclrtMemcpy(inTensor.deviceData,
        inTensor.dataSize,
        inTensorData.data(),
        sizeof(float) * inTensorData.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    atb::SVector<atb::Tensor> inTensors = {inTensor};
    return inTensors;
}
/**
 * @brief 创建一个Gelu Activation的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *GeluOperation()
{
    atb::infer::ActivationParam opParam;
    opParam.activationType = atb::infer::ActivationType::ACTIVATION_FASTER_GELU_FORWARD;
    atb::Operation *geluOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(opParam, &geluOp));
    return geluOp;
}
/**
 * @brief 创建一个Swiglu_forward Activation的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *SwigluOperation()
{
    atb::infer::ActivationParam opParam;
    opParam.activationType = atb::infer::ActivationType::ACTIVATION_SWIGLU_FORWARD;
    opParam.dim = -1;
    atb::Operation *swigluOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(opParam, &swigluOp));
    return swigluOp;
}
int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);
    // Activation Gelu示例
    atb::Operation *geluOp = GeluOperation();
    // 准备VariantPack
    atb::VariantPack geluVariantPack;
    geluVariantPack.inTensors = PrepareInTensor();  // 放入输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE, SEQ_LEN,
HIDDEN_SIZE});
```

 版权所有 © 华为技术有限公司

```
    geluVariantPack.outTensors.push_back(tensorOut);  // 放入输出tensor
    uint64_t geluWorkspaceSize = 0;
    // Gelu Activation准备工作
    CHECK_STATUS(geluOp->Setup(geluVariantPack, geluWorkspaceSize, context));
    uint8_t *geluWorkspacePtr = nullptr;
    if (geluWorkspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&geluWorkspacePtr), geluWorkspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // Gelu Activation执行
    geluOp->Execute(geluVariantPack, geluWorkspacePtr, geluWorkspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    for (atb::Tensor &inTensor : geluVariantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : geluVariantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (geluWorkspaceSize > 0) {
        CHECK_STATUS(aclrtFree(geluWorkspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(geluOp));  // operation，对象概念，先释放
    std::cout << "Gelu Activation demo success!" << std::endl;
    // Activation Swiglu_forward示例
    atb::Operation *swigluOp = SwigluOperation();
    // 准备VariantPack
    atb::VariantPack swigluVariantPack;
    swigluVariantPack.inTensors = PrepareInTensor();  // 放入输入tensor
    tensorOut = CreateTensor(ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE, SEQ_LEN,
HIDDEN_SIZE / 2});
    swigluVariantPack.outTensors.push_back(tensorOut);  // 放入输出tensor
    uint64_t swigluWorkspaceSize = 0;
    // Gelu Activation准备工作
    CHECK_STATUS(swigluOp->Setup(swigluVariantPack, swigluWorkspaceSize, context));
    uint8_t *swigluWorkspacePtr = nullptr;
    if (swigluWorkspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&swigluWorkspacePtr), swigluWorkspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // Gelu Activation执行
    swigluOp->Execute(swigluVariantPack, swigluWorkspacePtr, swigluWorkspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    for (atb::Tensor &inTensor : swigluVariantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : swigluVariantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (swigluWorkspaceSize > 0) {
        CHECK_STATUS(aclrtFree(swigluWorkspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(swigluOp));  // operation，对象概念，先释放
    std::cout << "Swiglu_forward Activation demo success!" << std::endl;
    // 资源释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    return 0;
}
```

10.8 AllGatherOperation

功能

将多个通信卡上的数据按所属rank号的顺序在第一维进行聚合，然后发送到每张卡上。

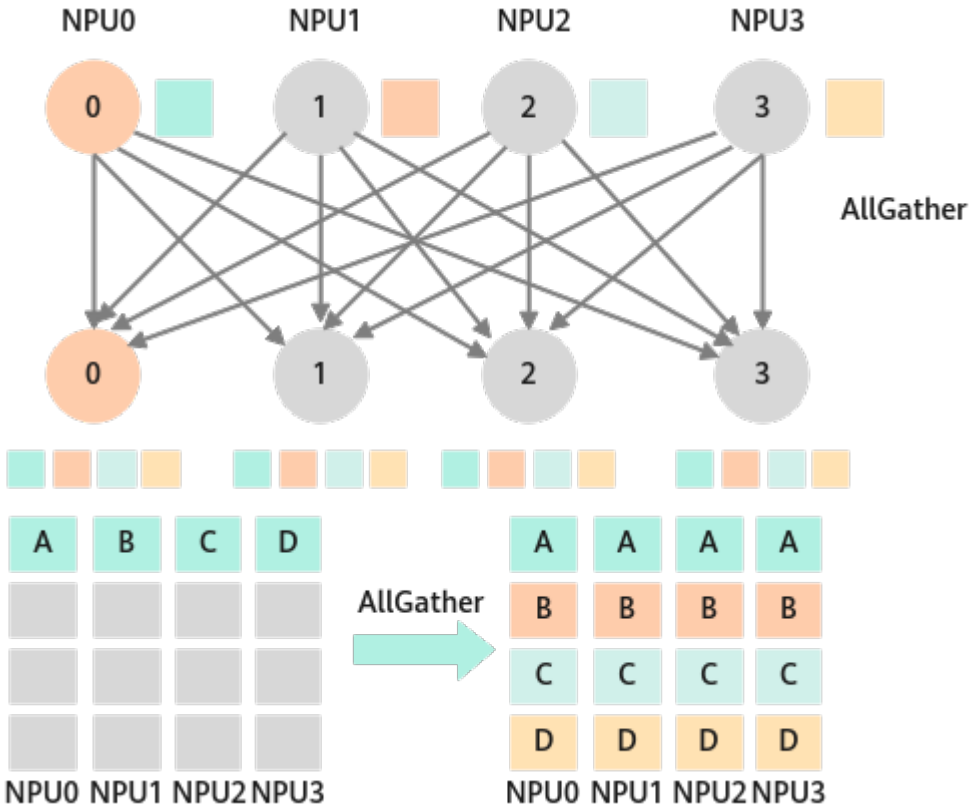
 版权所有 © 华为技术有限公司

该算子涉及多卡相关操作，可根据实际需求配置HCCL相关环境变量，具体请参见《CANN 环境变量参考》中的“**集合通信**”章节。

使用场景

多对多。收集所有的数据到所有的节点上。把多个节点的数据收集到一个主节点上（Gather），再把这个收集到的数据分发到其他节点上（broadcast）。

图 10-2 通信原理示意图



应用场景：

- All-Gather可应用于模型并行。
- 模型并行里前向计算里的参数全同步，需要用all-gather把模型并行里将切分到不同的NPU上的参数全同步到一张NPU上才能进行前向计算。

使用示例

- 参数设置

  rankSize=2
- 输入输出

  npu0（通信主卡）：

  x=

  ```
  [[16, 56],
  [ 5, 96]]
  ```

  output=

 版权所有 © 华为技术有限公司

```
[[[ 16,  56],
  [  5,  96]],

  [[ -7, -91],
  [-75,  38]]]
```

npu1：

x=

```
[[ -7, -91],
[-75,  38]]
```

output=

```
[[[ 16,  56],
  [  5,  96]],
[[ -7, -91],
[-75,  38]]]
```

定义

```
struct AllGatherParam {
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rank	int	0	当前卡所属通信编号。
rankSize	int	0	通信的卡的数量。
rankRoot	int	0	主通信编号。
backend	string	“hccl”	通信后端指示，仅支持“hccl”和“lccl”。 Atlas 推理系列产品仅支持“backend”为“hccl”。 当“backend”为“lccl”且机器为Atlas A2 训练系列产品里的Atlas 200T A2 Box16 异构子框时支持单机16卡（16卡全量拓扑通信或单节点内任意卡通信）。
hcclComm	HcclComm	nullptr	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	通信模式，**CommMode**类型枚举值。hccl多线程只支持外部传入通信域方式。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
rankTableFile	string	-	集群信息的配置文件路径，适用单机以及多机通信场景，当前仅支持hccl后端场景。 若单机配置了ranktable，则以ranktable来初始化通信域。 配置请参见《 TensorFlow 1.15模型迁移指南 》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。
commDomain	string	-	通信device组用通信域名标识，多通信域时使用。当backend为lccl时，commMode为多进程时，commDomain需要设置0-63。
rsv[64]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	• backend为hccl时：float16/float/int8/int16/int32/int64/bf16 • backend为lccl时：float16/float/int8/int16/int32/int64/bf16	ND	输入tensor，维度小于8。

输出

参数	维度	数据类型	格式	描述
output	[rankSize, dim_0，dim_1，... ，dim_n]	• backend为hccl时：float16/float/int8/int16/int32/int64/bf16 • backend为lccl时：float16/float/int8/int16/int32/int64/bf16	ND	输出tensor，维度小于或等于8。 输出output的维数比输入x的维数多一维。

规格约束

- rank、rankSize、rankRoot需满足以下条件。
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize

 版权所有 © 华为技术有限公司

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：
```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```
- 当前不支持单device上跑多个HCCP进程实例，不能单卡同时跑多个通信算子。比如运行通信算子场景下，不支持同一张卡上跑多个模型。

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <acl/acl.h>
#include <atb/atb_infer.h>
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>
#include "demo_util.h"
void ExcuteImpl(atb::Operation *op, atb::VariantPack variantPack, atb::Context *context)
{
    uint64_t workspaceSize = 0;
    CHECK_STATUS(op->Setup(variantPack, workspaceSize, context));
    void *workspace = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc(&workspace, workspaceSize, ACL_MEM_MALLOC_HUGE_FIRST));
    }
    CHECK_STATUS(op->Execute(variantPack, (uint8_t *)workspace, workspaceSize, context));
    if (workspace) {
        CHECK_STATUS(aclrtFree(workspace));  // 销毁workspace
    }
}
void AllGatherSample(int rank, int rankSize)
{
    int ret = aclInit(nullptr);
    // 设置每个进程对应的deviceId
    int deviceId = rank;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    aclrtStream stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);
    atb::Tensor input;
    input.desc.dtype = ACL_FLOAT16;
    input.desc.format = ACL_FORMAT_ND;
    input.desc.shape.dimNum = 2;
    input.desc.shape.dims[0] = 3;
    input.desc.shape.dims[1] = 5;
    input.dataSize = atb::Utils::GetTensorSize(input);
    CHECK_STATUS(aclrtMalloc(&input.deviceData, input.dataSize, ACL_MEM_MALLOC_HUGE_FIRST));
    atb::Tensor output;
    output.desc.dtype = ACL_FLOAT16;
    output.desc.format = ACL_FORMAT_ND;
    output.desc.shape.dimNum = 3;
    output.desc.shape.dims[0] = 2;
    output.desc.shape.dims[1] = 3;
    output.desc.shape.dims[2] = 5;
    output.dataSize = atb::Utils::GetTensorSize(output);
    CHECK_STATUS(aclrtMalloc(&output.deviceData, output.dataSize, ACL_MEM_MALLOC_HUGE_FIRST));
    atb::infer::AllGatherParam param;
```

 版权所有 © 华为技术有限公司

```
    param.rank = rank;
    param.rankRoot = 0;
    param.rankSize = rankSize;
    param.backend = "hccl";
    atb::Operation *op = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &op));
    atb::VariantPack variantPack;
    variantPack.inTensors = {input};
    variantPack.outTensors = {output};
    ExcuteImpl(op, variantPack, context);
    std::cout << "rank: " << rank << " executed END." << std::endl;
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(op));     // 销毁op对象
    CHECK_STATUS(aclrtDestroyStream(stream));    // 销毁stream
    CHECK_STATUS(atb::DestroyContext(context));  // 销毁context
    CHECK_STATUS(aclFinalize());
    std::cout << "demo excute success" << std::endl;
}
int main(int argc, const char *argv[])
{
    const int processCount = 2;
    for (int i = 0; i < processCount; i++) {
        pid_t pid = fork();
        // 子进程
        if (pid == 0) {
            AllGatherSample(i, processCount);
            return 0;
        } else if (pid < 0) {
            std::cerr << "Failed to create process." << std::endl;
            return 1;
        }
    }
    // 父进程等待子进程执行完成
    for (int i = 0; i < processCount; ++i) {
        wait(NULL);
    }
    std::cout << "The communication operator is successfully executed. Parent process exit" << std::endl;
    return 0;
}
```

10.9 AllGatherVOperation

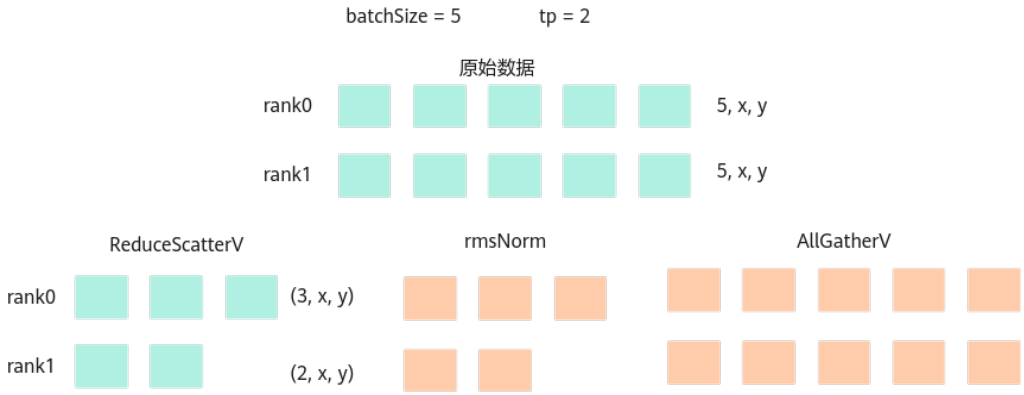
功能

将多个通信卡上的数据按通信编号的顺序在第一维进行聚合，然后发送到每张卡上。支持每张卡发送的数据不等长。

推理场景中会出现batch size不能被TP数整除的情况，reducescatter后续的计算算子需要按照batch维度处理数据，再将处理数据进行allgather，如图**图10-3**所示。

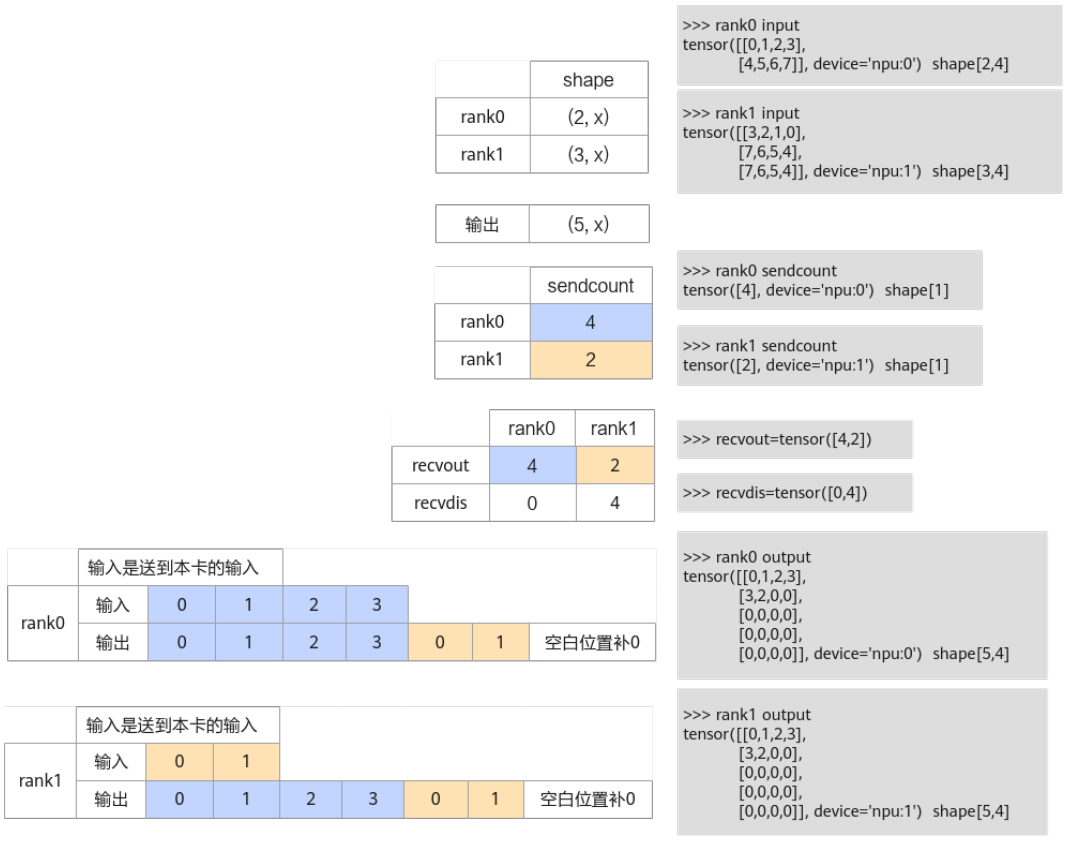
 版权所有 © 华为技术有限公司

图 10-3 算子上文示意图



示例：

图 10-4 计算过程示意图



计算过程示意（python）：

```
# 计算goldtensor
gold_outtensor = []
for i in range(len(sendcount)):
    gold_outtensor= gold_outtensor+(tensorafters[i][0:sendcount[i]])
GoldenTensors = (torch.tensor(np.array(gold_outtensor+[0]*(sum*dim[1]-
len(gold_outtensor))).reshape(sum,dim[1]), dtype=inTensorDtype))
```

 版权所有 © 华为技术有限公司

使用场景

模型并行；模型并行里前向计算里的参数全同步，需要用AllgatherV把模型并行里将切分到不同的XPU上的参数全同步到一张XPU上才能进行前向计算。

```
>>> rank0 input
tensor([[0,1,2,3],
        [4,5,6,7]], device='npu:0')  shape[2,4]
>>> rank0 sendcount
tensor([4], device='npu:0')  shape[1]
>>> rank1 input
tensor([[3,2,1,0],
        [7,6,5,4],
        [7,6,5,4]], device='npu:1')  shape[3,4]
>>> rank1 sendcount
tensor([2], device='npu:1')  shape[1]
>>> recvout=tensor([4,2])
>>> recvdis=tensor([0,4])
>>> y=tensor([0,1,2,3,4], device='npu:0')
>>> rank0 output
tensor([[0,1,2,3],
        [3,2,0,0],
        [0,0,0,0],
        [0,0,0,0],
        [0,0,0,0]], device='npu:0')  shape[5,4]
>>> rank1 output
tensor([[0,1,2,3],
        [3,2,0,0],
        [0,0,0,0],
        [0,0,0,0]], device='npu:1')  shape[5,4]
```

定义

```
struct AllGatherVParam {
    int rank = -1;
    int rankSize = 0;
    int rankRoot = 0;
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rank	int	-1	当前卡所属通信编号。-1表示未传。
rankSize	int	0	通信的卡的数量，不能为0。
rankRoot	int	0	主通信编号。
backend	std::string	"hccl"	通信计算类型，仅支持"hccl"。
hcclComm	HcclComm	nullptr	hccl通信域接口获取的地址指针。默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
commMode	CommMode	COMM_MULTI_PROCESS	通信模式，CommMode类型枚举值。hccl多线程只支持外部传入通信域方式。 • COMM_UNDEFINED：未定义。 • COMM_MULTI_PROCESS：指定多进程通信。 • COMM_MULTI_THREAD： 指定多线程通信。
rankTableFile	std::string	-	集群信息的配置文件路径，用于多机通信场景，当前仅支持hccl后端场景。集群信息的配置文件路径，适用单机以及多机通信场景，当前仅支持hccl后端场景，若单机配置了rankTable，则以ranktable来初始化通信域。 ranktable配置请参考：《 TensorFlow 1.15模型迁移指南 》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。
commDomain	std::string	-	通信device组用通信域名标识，多通信域时使用，当前仅支持hccl。
rsv[64]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	是否必选	描述
x	[dim_0, dim_1, ..., dim_n]	"hccl": float16/ int8/ bfloat16	ND	是	输入tensor。
sendCount	1[1]	int64	ND	是	输入tensor，为本卡发送的数据量。支持每张卡的该tensor不同，即支持数据不等长。
recvCounts	1[ranksize]	int64	ND	是	输入tensor，为从对应索引卡号接收到的数据量，每张卡都一样。
rdispls	1[ranksize]	int64	ND	是	为从对应索引卡号接收到的数据量的偏移，每张卡都一样，rdispls[i] = n表示本rank从相对于输入起始位置的偏移量为n的位置开始接收rank_i的数据。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	是否必选	描述
y	1[每个卡的第1维的shape的和]	float16	ND	是	shape为所有卡的合并tensor的首shape之和，用于infer shape。

输出

参数	维度	数据类型	格式	是否必选	描述
output	[n, dim_1, ..., dim_n]	"hccl": float16/ int8/ bfloat16	ND	否	输出tensor，与输入tensor地址不同，非原地写，n为所有卡的要合并tensor的第一维shape之和，即y的shape。数据类型和输入相同。

规格约束

- backend必须是hccl。
- 数组参数recvCounts、rdispls的长度等于ranksize。
- 卡rank的sendCount与recvCounts[rank]相等。
- out tensor第1维的shape等于y的shape。
- 仅支持Atlas 推理系列产品和Atlas 800I A2 推理产品。
- 各个通信卡的输入tensor的第0维维度可不同，其它维维度需相同。
- recvCounts数组元素之和不能溢出int64，recvCounts的数组和要大于0。
- sendCount的值不能超过输入tensor x的所有维度的shape的乘积。
- rank、rankSize、rankRoot需满足以下条件。
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize

接口调用示例（ Python ）

```
def main_worker(rank, world_size, inTensorDtypes, random_seed):
    torch_npu.npu.set_device(rank)
    print(f'Process {rank} started, using device npu:{rank}.')
    acl_allgatherv_operation = torch.classes.OperationTorch.OperationTorch(
        "AllGatherVOperation")
    torch.manual_seed(random_seed)
    for inTensorDtype in inTensorDtypes:
        acl_param = json.dumps({"rank": rank, "rankSize": world_size, "rankRoot": 0, "backend": "hccl"})
        acl_allgatherv_operation.set_param(acl_param)

        op_name = "AllGatherVOperation"
        run_param = json.dumps({"sendCount": sendcount[rank], "recvCounts": recvout[rank], "rdispls":
recvdis[rank]})
        host_list = [[sendcount[rank]], recvout[rank], recvdis[rank]]
        host_tensors = [np.array(tensor) for tensor in host_list]
```

 版权所有 © 华为技术有限公司

```
        host_tensors = [torch.from_numpy(tensor).to(torch.int64) for tensor in host_tensors]
        host_tensors = [tensor.npu() for tensor in host_tensors]
        acl_allgatherv_operation.set_varaintpack_param(run_param)

        inTensors = [torch.tensor([[0, 1, 2, 3], [4, 5, 6, 7]], dtype=inTensorDtype),
                     torch.tensor([[3, 2, 1, 0], [7, 6, 5, 4], [7, 6, 5, 4]], dtype=inTensorDtype)]
        # y用来推导outputshape，长度应为所有inputtensor的dim0之和
        y = torch.tensor([0, 1, 2, 3, 4], dtype=torch.float16)
        GoldenTensors = [
            torch.tensor([[0, 1, 2, 3], [3, 2, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]], dtype=inTensorDtype),
            torch.tensor([[0, 1, 2, 3], [3, 2, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]],
                         dtype=inTensorDtype)]
        acl_out_tensor = acl_allgatherv_operation.execute([inTensors[rank].npu(), host_tensors[0],
                                                           host_tensors[1], host_tensors[2], y.npu()])[0]
        print(f"********************acl_out_tensor:{acl_out_tensor}")
        torch.npu.synchronize()
        # assert result
        assert golden_compare(GoldenTensors[rank], acl_out_tensor.cpu())


def test_all_gather(self):
    world_size = 2
    random_seed = 123
    inTensorDtypes = [torch.int8]
    set_start_method('spawn', force=True)
    process_list = []
    for i in range(world_size):
        p = Process(target=main_worker, args=(i, world_size, inTensorDtypes, random_seed))
        p.start()
        process_list.append(p)
    for i in process_list:
        p.join()
```

10.10 AllReduceOperation

功能

将多个通信卡上的数据进行计算，支持相加、取最大、最小、相乘四种计算，然后发送到每张卡上。

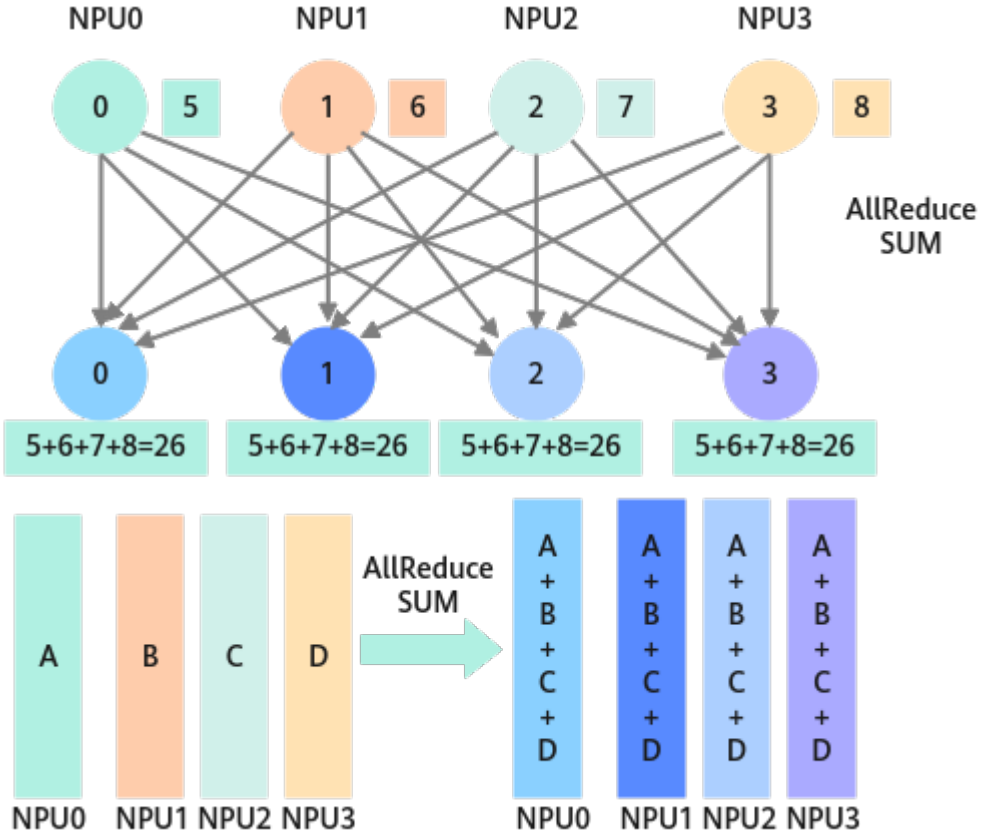
该算子涉及多卡相关操作，可根据实际需求配置HCCL相关环境变量，具体请参见《CANN 环境变量参考》中的“**集合通信**”章节。

使用场景

多对多。在所有节点上都执行相同的Reduce操作（相加、取最大、最小、相乘），将所有节点的数据规约运算得到的结果发送到所有的节点上（broadcast）。

 版权所有 © 华为技术有限公司

图 10-5 AllReduce 算子工作原理



应用场景：

- AllReduce里的 broadcast + reduce组合里的reduce操作。
- ReduceScatter组合里的 reduce操作。
- 分布式训练parameter server 参数服务器结构里的 master节点 broadcast 数据到worker节点，再从worker节点reduce数据回master节点里的reduce操作。

使用示例：

- **非量化场景 (rankSize=2):**
  - 场景1：allReduce sum

    npu0:

    ```
    x=[-4,-50]
    output=[-8,-100]
    ```

    npu1:

    ```
    x=[-4,-50]
    output=[-8,-100]
    ```

  - 场景2：allReduce max

    npu0:

    ```
    x=[4,20]
    output=[10,50]
    ```

    npu1:

    ```
    x=[10,50]
    output=[10,50]
    ```

 版权所有 © 华为技术有限公司

  - 场景3：allReduce min

    npu0:

    ```
    x=[3,100]
    output=[3,20]
    ```

    npu1:

    ```
    x=[3,20]
    output=[3,20]
    ```

  - 场景4：allReduce prod

    npu0:

    ```
    x=[-93,-48]
    output=[8649,2304]
    ```

    npu1:

    ```
    x=[-93,-48]
    output=[8649,2304]
    ```

- **量化场景：allReduce sum时进行量化传输（rankSize=2）：**
  - 量化计算公式：

    $$output = (allReduce(input) + offset * ranksize) * scale$$

  - 场景1：QUANT_TYPE_PER_TENSOR 对整个张量进行量化。

    ```
    x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
    scale=[1]
    offset=[0]
    output=[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32]
    ```

  - 场景2：QUANT_TYPE_PER_CHANNEL 对张量中的每个channel分别进行量化。

    ```
    x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
    scale=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
    offset=[0]
    output=[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32]
    ```

定义

```
struct AllReduceParam {
    enum QuantType : int {
        QUANT_TYPE_UNQUANT = 0,
        QUANT_TYPE_UNDEFINED = 0,
        QUANT_TYPE_PER_TENSOR = 1,
        QUANT_TYPE_PER_CHANNEL = 2,
        QUANT_TYPE_MAX = 3,
    };
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    std::string allReduceType = "sum";
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    std::string commDomain;
    QuantType quantType = QUANT_TYPE_UNDEFINED;
    aclDataType outDataType = ACL_DT_UNDEFINED;
    uint8_t rsv[64] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
rank	int	0	[0, rankSize-1]	是	当前卡所属通信编号。
rankSize	int	0	-	是	通信的卡的数量。
rankRoot	int	0	[0, rankSize-1]	是	主通信编号。
allReduceType	string	"sum"	sum prod max min	是	通信计算类型。 支持“sum”（相加），“prod”（相乘），“max”（取最大）和“min”（取最小）。
backend	string	"hccl"	lccl/hccl	是	通信计算类型，仅支持“hccl”和“lccl”。Atlas 推理系列产品仅支持backend为"hccl"。 ● “backend”为“hccl”时 – 支持"sum"，"prod"，"max"和"min"。 – Atlas 800I A2 推理产品/Atlas A2 训练系列产品上“allReduceType”为"prod"时，不支持数据类型为int16和bf16。 – Atlas 推理系列产品不支持int64、bf16，int16只有“allReduceType”为"sum"时支持。 – Atlas 训练系列产品不支持int16、bf16。 – 不支持量化。 ● “backend”为“lccl”时 – 支持"sum"，"max"和"min"。 – 不支持数据类型int64。 – 机器为Atlas A2 训练系列产品里的Atlas 200T A2 Box16 异构子框时支持单机16卡（16卡全量拓扑通信或单节点内任意卡通信）。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
hcclComm	HcclComm	nullptr	-	否	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	• COMM_MULTI_PROCESS • COMM_MULTI_THREAD	否	通信模式，**CommMode**类型枚举值。hccl多线程仅支持外部传入通信域方式。
rankTableFile	string	-	-	否	集群信息的配置文件路径，用于单机以及多机通信场景，当前仅支持HCCL后端场景。 若单机配置了ranktable，则以ranktable来初始化通信域。 配置请参见《TensorFlow 1.15模型迁移指南》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。
commDomain	string	-	-	否	通信Device组用通信域名标识，多通信域时使用。当backend为"lccl"时，commMode为多进程时，commDomain需要设置为0-63的数字。commMode为多线程时，不支持确定性计算，"LCCL_DETERMINISTIC"需要为0或者false。 LCCL在多进程/多线程多通信域并发场景下，"LCCL_PARALLEL"需要设置为1或者true。多通信域并行功能使用结束后，"LCCL_PARALLEL"需要设置为0或者false，否则会导致基础场景性能下降。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
quantType	QuantType	QUANT_TYPE_UNQUANT	• QUANT_TYPE_UNQUANT • QUANT_TYPE_UNDEFINED • QUANT_TYPE_PER_TENSOR • QUANT_TYPE_PER_CHANNEL • QUANT_TYPE_MAX	否	量化类型。 QUANT_TYPE_UNQUANT：默认值。 QUANT_TYPE_UNDEFINED：默认值。 QUANT_TYPE_PER_TENSOR：对整个张量进行量化。 QUANT_TYPE_PER_CHANNEL：对张量中每个channel分别进行量化。 QUANT_TYPE_MAX：枚举类型最大值。
outDataType	aclDataType	ACL_DT_UNDEFINED	ACL_FLOAT16	否	• 若为浮点AllReduce，参数outDataType配置为ACL_DT_UNDEFINED，表示输出tensor的数据类型与输入tensor一致。 • 若为量化AllReduce，输出tensor的数据类型与输入tensor不一致，则参数outDataType配置为用户预期输出tensor的数据类型。量化只支持配置ACL_FLOAT16。
rsv[64]	uint8_t	{0}	[0]	否	预留参数。

注：LCCL多通信域并行时，最大通信数据量不超过100MB，1MB以上性能下降。并发通信时，带宽共享，性能下降。LCCL多通信域并行功能和确定性计算功能不兼容，开启确定性计算后该功能失效。

 版权所有 © 华为技术有限公司

函数输入输出描述

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	• "hccl": float16/ float/int8/ int16/ int32/ int64/bf16 • "lccl": float16/ float/int8/ int16/ int32/bf16	ND	输入tensor。
output	[dim_0, dim_1, ..., dim_n]	• "hccl": float16/ float/int8/ int16/ int32/ int64/bf16 • "lccl": float16/ float/int8/ int16/ int32/bf16	ND	输出tensor。

quantType 为 QUANT_PER_Tensor 的输入输出描述

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	int8	ND	输入tensor，最后一维n的大小是16的整数倍。
scale	[1]	float16	ND	输入。scale中元素要求为标量。
offset	[1]	float16	ND	输入。offset中元素要求为标量。
y	[dim_0, dim_1, ..., dim_n]	float16	ND	量化输出结果；x和y的shape一致。

 版权所有 © 华为技术有限公司

quantType 为 QUANT_PER_Channel 的输入输出描述

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	int8	ND	输入tensor，最后一维n的大小是16的整数倍。
scale	[1, n] 或 [n]	float16	ND	输入。scale中元素要求不为0。最后一维n的大小是16的整数倍。
offset	[1]	float16	ND	输入。offset中元素要求为标量。
y	[dim_0, dim_1, ..., dim_n]	float16	ND	量化输出结果；x和y的shape一致。

规格约束

rank、rankSize、rankRoot需满足以下条件。

- 0 ≤ rank < rankSize
- 0 ≤ rankRoot < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：

```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```

- 当前不支持单device上跑多个HCCP进程实例，不能单卡同时跑多个通信算子。比如运行通信算子场景下，不支持同一张卡上跑多个模型。

算子调用示例（C++）

前置条件和编译命令请参见4 算子调用示例。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：基础场景。

说明

因为用例使用多线程，编译时要增加编译选项：

```
-l pthread
```

```
#include <iostream>
#include <vector>
#include <numeric>
#include <string>
#include <future>
#include "acl/acl.h"
#include "atb/operation.h"
```

 版权所有 © 华为技术有限公司

```
#include "atb/types.h"
#include "atb/atb_infer.h"
#include "demo_util.h"

struct Args {
    int rankId;
    aclrtStream stream;
    atb::Context *context;
    atb::infer::AllReduceParam param;
};

/**
 * @brief 准备atb::VariantPack中的所有tensor
 * @param args 线程参数结构体
 * @return atb::VariantPack
 */
atb::VariantPack PrepareVariantPack(Args &args)
{
    atb::VariantPack variantPack;
    std::vector<int64_t> shape = {2, 1024};
    // 创建Host侧数据
    std::vector<float> xHostData(shape[0] * shape[1], 2.0);
    std::vector<float> outputHostData(shape[0] * shape[1], 0);
    // 生成ATB tensor
    atb::Tensor tensorX =
        CreateTensorFromVector(args.context, args.stream, xHostData, ACL_FLOAT16,
aclFormat::ACL_FORMAT_ND, shape);
    atb::Tensor tensorOutput =
        CreateTensorFromVector(args.context, args.stream, outputHostData, ACL_FLOAT16,
aclFormat::ACL_FORMAT_ND, shape);
    // 根据顺序将所有输入tensor放入SVector
    variantPack.inTensors = {tensorX};
    variantPack.outTensors = {tensorOutput};
    return variantPack;
}

/**
 * @brief 创建一个AllReduce的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
int RunAllReduceOp(Args &args)
{
    int rankId = args.rankId;
    auto aclRet = aclrtSetDevice(rankId);
    CHECK_STATUS_EXPR(aclRet, std::cout << "aclrtSetDevice failed" << std::endl; return aclRet);
    atb::VariantPack variantPack = PrepareVariantPack(args);
    atb::Operation *allReduceOp = nullptr;
    atb::Status st = atb::CreateOperation(args.param, &allReduceOp);
    CHECK_STATUS_EXPR(
        (st != atb::ErrorType::NO_ERROR || !allReduceOp), std::cout << "atb create operation failed" << std::endl;
        return st);
    uint64_t workspaceSize = 0;
    // ATB Operation 第一阶段接口调用：对输入输出进行检查，并根据需要计算workspace大小
    st = allReduceOp->Setup(variantPack, workspaceSize, args.context);
    CHECK_STATUS_EXPR(st, std::cout << "operation setup failed" << std::endl; return st);
    // 根据第一阶段接口计算出的workspaceSize申请device内存
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        aclRet = aclrtMalloc((void **)(&workspacePtr), workspaceSize, ACL_MEM_MALLOC_HUGE_FIRST);
        CHECK_STATUS_EXPR(aclRet, std::cout << "aclrtMalloc failed" << std::endl; return aclRet);
    }
    // ATB Operation 第二阶段接口调用：执行算子
    st = allReduceOp->Execute(variantPack, workspacePtr, workspaceSize, args.context);
    CHECK_STATUS_EXPR(st, std::cout << "operation execute failed" << std::endl; return aclRet);
    // 同步等待
    aclRet = aclrtSynchronizeStream(args.stream);
    CHECK_STATUS_EXPR(aclRet, std::cout << "aclrtSynchronizeStreame failed" << std::endl; return aclRet);
    // 资源释放
    for (atb::Tensor &tensor : variantPack.inTensors) {
```

 版权所有 © 华为技术有限公司

```
        if (tensor.deviceData) {
            aclrtFree(tensor.deviceData);
        }
    }
    for (atb::Tensor &tensor : variantPack.outTensors) {
        if (tensor.deviceData) {
            aclrtFree(tensor.deviceData);
        }
    }
    if (workspaceSize > 0 && workspacePtr != nullptr) {
        aclrtFree(workspacePtr);
    }
    st = atb::DestroyOperation(allReduceOp);
    CHECK_STATUS_EXPR(st, std::cout << "destroy operation failed" << std::endl; return st);
    aclRet = aclrtDestroyStream(args.stream);
    CHECK_STATUS_EXPR(aclRet, std::cout << "aclrtDestroyStream failed" << std::endl; return aclRet);
    st = atb::DestroyContext(args.context);
    CHECK_STATUS_EXPR(st, std::cout << "destroy context failed" << std::endl; return st);
    aclRet = aclrtResetDevice(args.rankId);
    CHECK_STATUS_EXPR(aclRet, std::cout << "aclrtResetDevice failed" << std::endl; return aclRet);
    return 0;
}

int main(int argc, char **argv)
{
    // 设置通信设备数量
    static const int DEV_NUM = 8;
    // 设置通信计算类型
    static const std::string ALL_REDUCE_TYPE = "sum";
    // 设置通信计算后端
    static const std::string BACKEND = "lccl";
    CHECK_STATUS(aclInit(nullptr));
    Args args[DEV_NUM];
    for (int rankId = 0; rankId < DEV_NUM; rankId++) {
        args[rankId].rankId = rankId;
        // 设置当前设备
        CHECK_STATUS(aclrtSetDevice(rankId));
        // 创建ATB Context
        CHECK_STATUS(atb::CreateContext(&(args[rankId].context)));
        // 创建流
        CHECK_STATUS(aclrtCreateStream(&(args[rankId].stream)));
        args[rankId].context->SetExecuteStream(args[rankId].stream);
        // 配置AllReduce Param
        atb::infer::AllReduceParam &param = args[rankId].param;
        param.rank = rankId;
        param.rankSize = DEV_NUM;
        param.allReduceType = ALL_REDUCE_TYPE;
        param.backend = BACKEND;
        param.commMode = atb::infer::CommMode::COMM_MULTI_THREAD;
        param.commDomain = "domain0";  // 单通信域demo
    }
    // 启动多线程任务
    std::vector<std::future<int>> threads(DEV_NUM);
    for (int rankId = 0; rankId < DEV_NUM; rankId++) {
        threads[rankId] = std::async(&RunAllReduceOp, std::ref(args[rankId]));
    }
    for (int rankId = 0; rankId < DEV_NUM; rankId++) {
        int aclRet = threads[rankId].get();
        CHECK_STATUS_EXPR(aclRet, std::cout << "multithread task " << rankId << " failed" << std::endl);
    }
    CHECK_STATUS(aclFinalize());
    std::cout << "AllReduce demo finished!" << std::endl;
    return 0;
}
```

 版权所有 © 华为技术有限公司

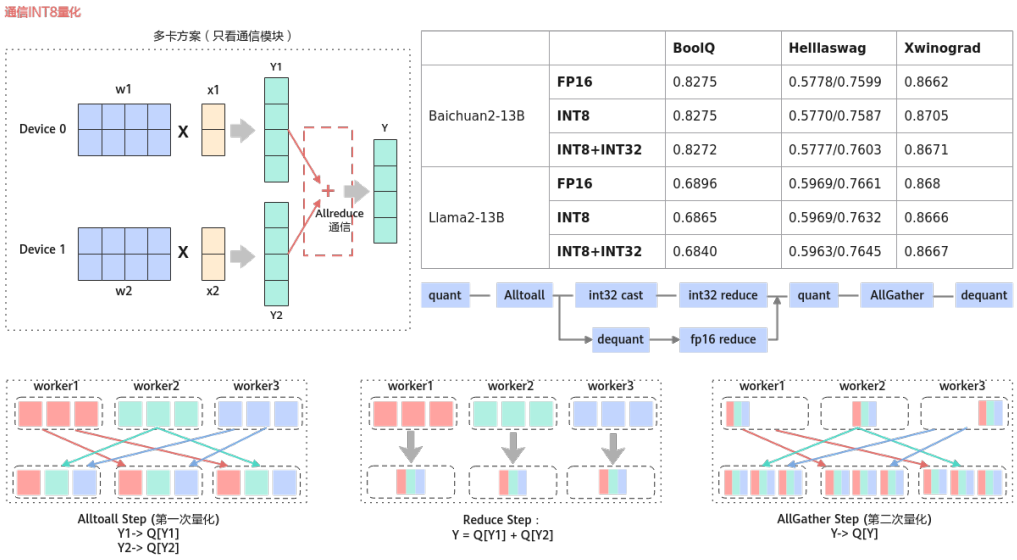
10.11 AllToAllOperation

功能

向通信域内所有通信卡发送相同数据量（输入切分成rankSize份）的数据，并从所有通信卡接收相同数据量的数据。

算子上下文

图 10-6 通信 INT8 量化中 AllToAll 的位置



		BoolQ	Helllaswag	Xwinograd
Baichuan2-13B	**FP16**	0.8275	0.5778/0.7599	0.8662
	INT8	0.8275	0.5770/0.7587	0.8705
	INT8+INT32	0.8272	0.5777/0.7603	0.8671
Llama2-13B	**FP16**	0.6896	0.5969/0.7661	0.868
	INT8	0.6865	0.5969/0.7632	0.8666
	INT8+INT32	0.6840	0.5963/0.7645	0.8667

模型传输一个int8的量化输入tensor数据X，首先使用AlltoAll进行通信发送到各个节点上（这里使用int8进行通信，提升了通信速度）。

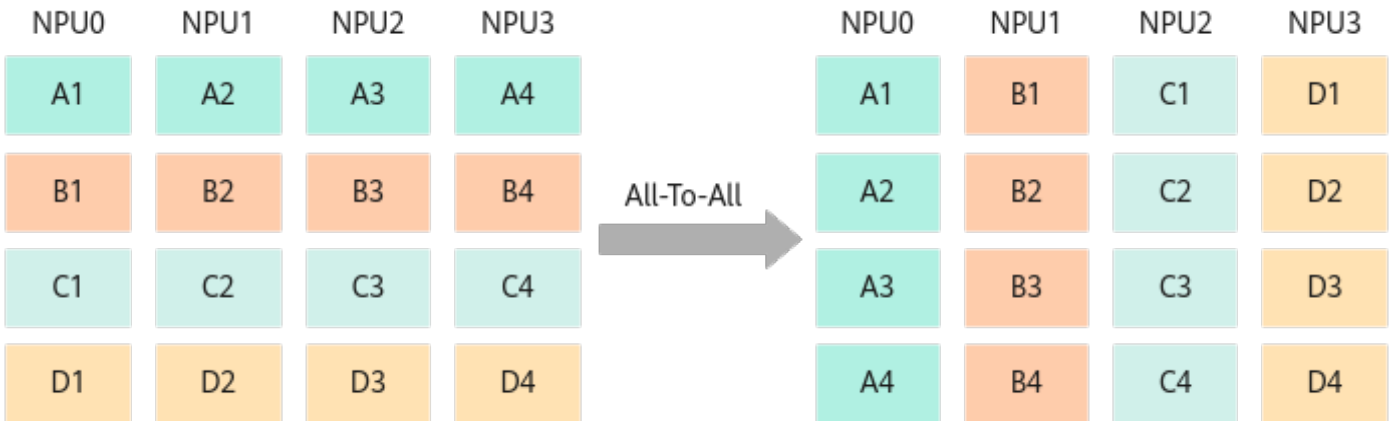
然后使用reduce的sum操作对x进行求和，将int8数据反量化为float16，最后使用AllGather进行通信将计算结果传输到各个节点上。（需求图中AllGather前后的量化，有可能会有精度损失，当前未实现。）

算子功能实现描述

向通信域内所有通信卡发送相同数据量（输入切分成rankSize份）的数据，并从所有通信卡接收相同数据量的数据。如下图所示，对于rankSize=4的情况，各个通信卡将自身的数据均匀分成rankSize份，发送给其他的通信卡。

 版权所有 © 华为技术有限公司

图 10-7 AllToAll 算子功能示意图



计算过程示意（Python）：

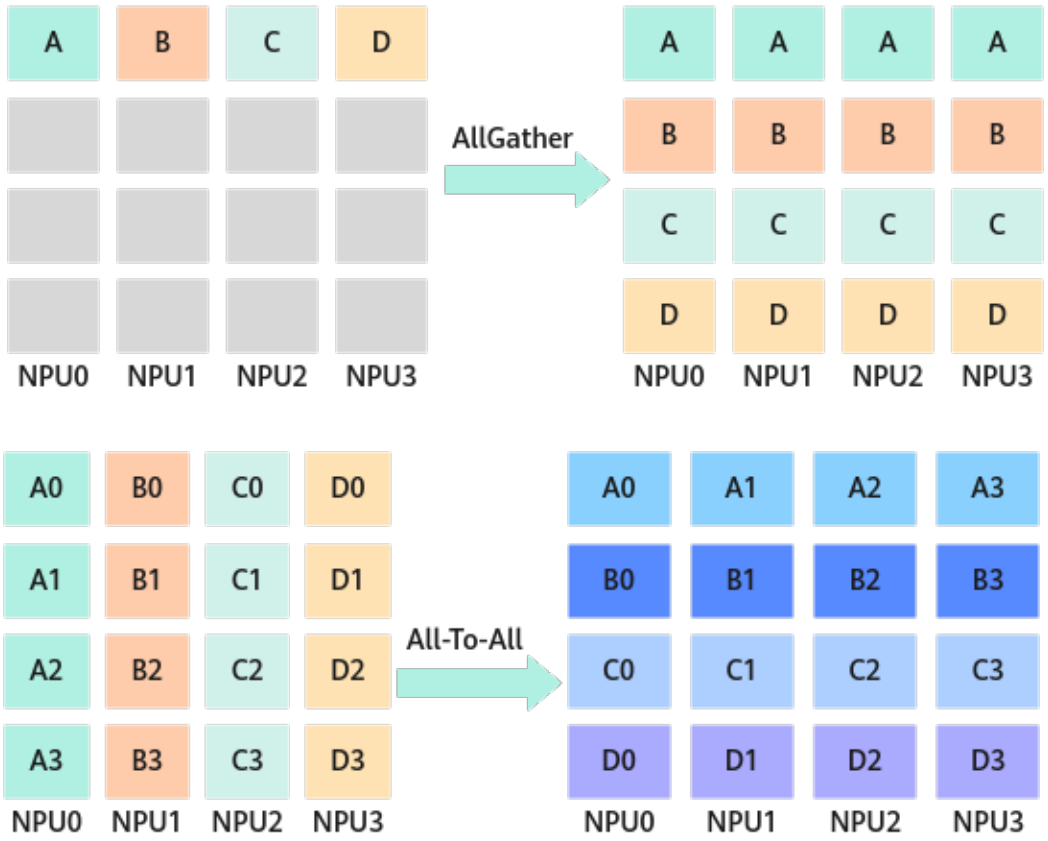
```
goldenTensors = []
for i in range(rankSize):
    golden_out = []
    for j in range(rankSize):
        golden_out_list = intensors[j].reshape(-1).tolist() # intensors[j]代表第j张卡上的intesor，计算时视为一维向量
        split = golden_out_list[i*len(golden_out_list) // rankSize:(i+1)*len(golden_out_list) // rankSize]
        golden_out += split
    golden_out_tensor = torch.tensor(golden_out).reshape(shape) # golden_out_tensor是第i张卡上的输出tensor，输出时shape和输入tensor一致
    goldenTensors.append(golden_out_tensor)
return [goldenTensors[rank].cpu()]
```

使用场景

用于将数据发送到各个节点上，多对多。AllToAll是对Allgather的扩展，相比于AllGather，AllToAll不同的节点从某一节点收集到的数据是不同的。

 版权所有 © 华为技术有限公司

图 10-8 AllToAll 和 AllGather 使用对比



应用于模型并行；模型并行里的矩阵转置；数据并行到模型并行的矩阵转置。

定义

```
struct AllToAllParam {
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    std::string commDomain;
    bool transpose = false;
    uint8_t rsv[62] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rank	int	0	当前卡所属通信编号。
rankSize	int	0	通信的卡的数量。
rankRoot	int	0	主通信编号

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
backend	std::string	"hccl"	通信计算类型，支持hccl和lccl，Atlas 800I A2 推理产品/Atlas A2 训练系列产品上为lccl时，transpose只能为true。
hcclComm	HcclComm	nullptr	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	通信模式，**CommMode**类型枚举值。hccl多线程只支持外部传入通信域方式。
rankTableFile	std::string	-	集群信息的配置文件路径，适用单机以及多机通信场景，当前仅支持hccl后端场景。 若单机配置了rankTable，则以rankTable来初始化通信域。 配置请参见《TensorFlow 1.15模型迁移指南》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。
commDomain	std::string	-	通信device组用通信域名标识，多通信域时使用，该算子无需传此参数。
transpose	bool	false	通信结果对输入进行转置，当前仅支持lccl。
rsv[62]	uint8_t	{0}	预留参数。

基础场景输入输出

transpose置为false

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/float/int8/int16/int32/int64/bf16	ND	输入tensor。
output	[dim_0, dim_1, ..., dim_n]	float16/float/int8/int16/int32/int64/bf16 数据类型和输入相同	ND	输出tensor，与输入维度相同。

转置场景输入输出

transpose置为true

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
x	[b, w]	float16/bf16/int8	ND	输入tensor。
output	[b*rankSize, w/rankSize]	float16/bf16/int8	ND	输出tensor。

规格约束

- 当前只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- backend为lccl时不支持多机通信和多通信域，且只支持偶数的rankSize。
- 各个通信卡的输入tensor维度相等。
- 参数维度相乘要能整除rankSize。
- 进行AllToAll计算时会将输入tensor视为一维向量计算，输出的shape则会和输入tensor一致。
- 转置场景下有如下约束：
  - w需要被rankSize整除。
  - w/rankSize的值不能超过90KB。
  - 整个x tensor的大小不能超过190MB。
  - rankSize只能为2、4、8。
- rank、rankSize、rankRoot需满足以下条件。
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：

```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```

10.12 AllToAllVOperation

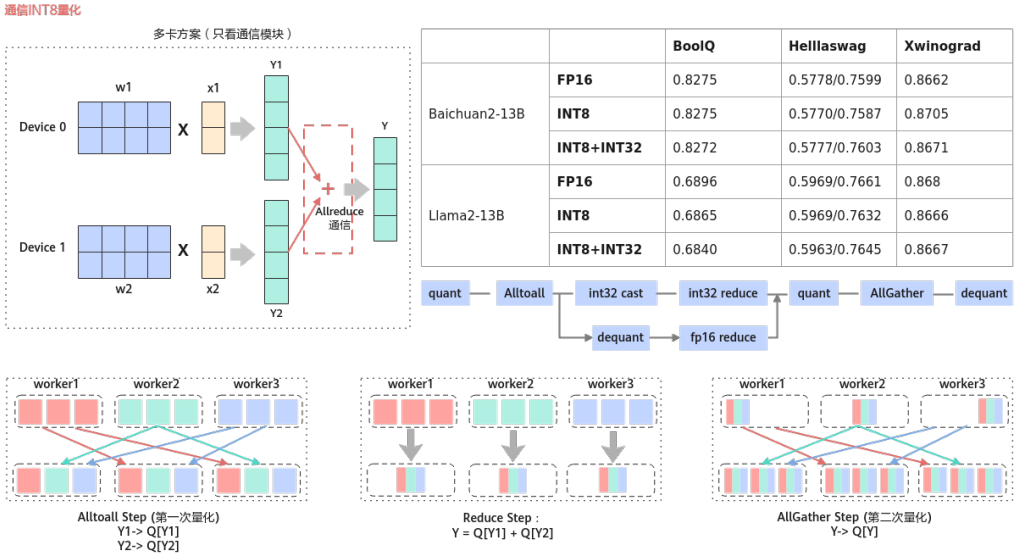
功能

向通信域内所有通信卡发送数据（数据量可以定制），并从所有通信卡接收数据。

 版权所有 © 华为技术有限公司

算子上下文

图 10-9 通信 INT8 量化中 AllToAll 的位置



		BoolQ	Hellaswag	Xwinograd
Baichuan2-13B	**FP16**	0.8275	0.5778/0.7599	0.8662
	INT8	0.8275	0.5770/0.7587	0.8705
	INT8+INT32	0.8272	0.5777/0.7603	0.8671
Llama2-13B	**FP16**	0.6896	0.5969/0.7661	0.868
	INT8	0.6865	0.5969/0.7632	0.8666
	INT8+INT32	0.6840	0.5963/0.7645	0.8667

模型传输一个int8的量化输入tensor数据X,首先使用AlltoAll进行通信发送到各个节点上（这里使用int8进行通信，提升了通信速度）

然后使用reduce的sum操作对x进行求和，将int8数据反量化为float16, 最后使用AllGather进行通信将计算结果传输到各个节点上。

算子功能实现描述

向通信域内所有通信卡发送数据（数据量可以通过参数定制），并从所有通信卡接收数据（数据量可以通过参数定制）。

- **示例**

 版权所有 © 华为技术有限公司

图 10-10 算子输入输出示例

		rank0	rank1	rank2
rank0	sendcount	2	4	2
	senddisp	0	2	6
	recvout	2	2	1
	recvdisp	0	2	4

rank0 输入要送到所有的卡				
输入	0	1	2	
输出	0	1	0	

rank1	sendcount	2	1	2
	senddisp	0	2	3
	recvout	4	1	1
	recvdisp	0	4	5

rank1 输入要送到所有的卡				
输入	0	1	2	
输出	2	3	4	

rank2	sendcount	1	1	1
	senddisp	0	1	2
	recvout	2	2	1
	recvdisp	0	2	4

rank2 输入要送到所有的卡				
输入	0	1	2	
输出	6	7	3	

- **计算过程python示意**

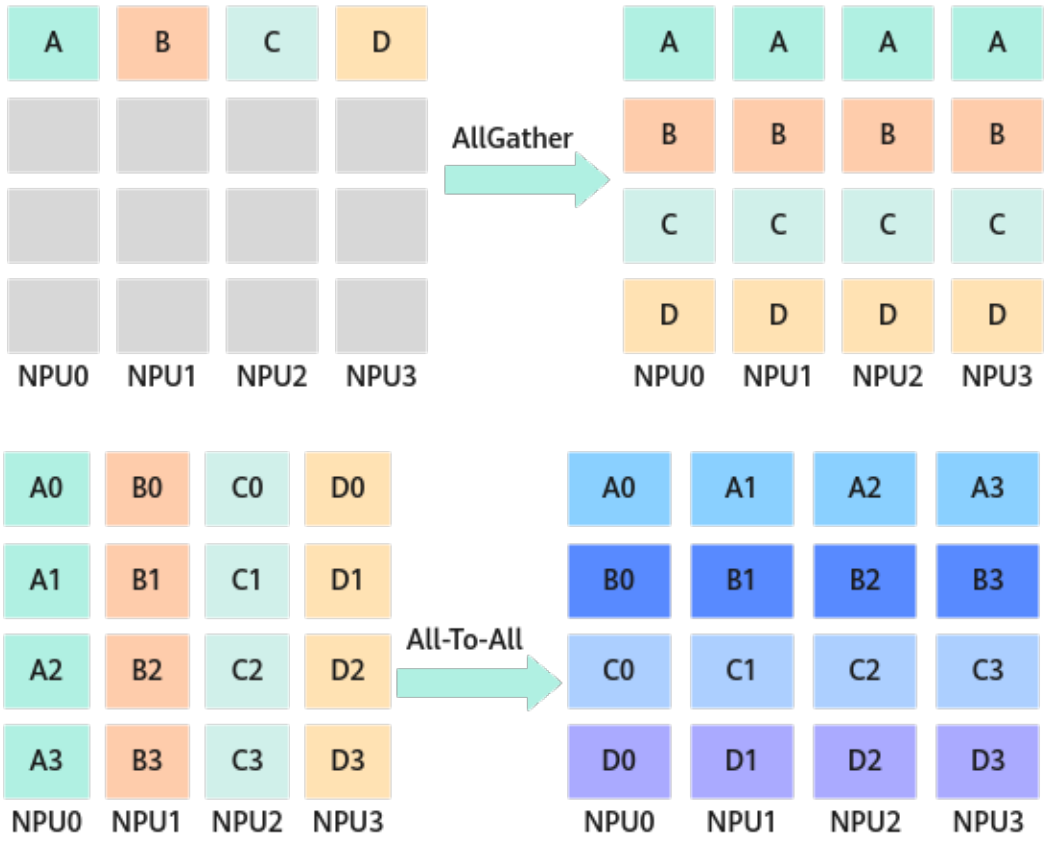
```
golden_result = []
for j in range(rankSize):
    tensor_list = intensors[j].reshape(-1).tolist() # intensors[j]代表第j张卡上的intensor
    golden_result += tensor_list[sdispls[rank]:sdispls[rank]+sendCounts[rank]]
size = sum(recvCounts)
golden_out_tensor = torch.tensor(golden_result).reshape(1,size)
```

使用场景

用于将数据发送到各个节点上，多对多。对All-Gather的扩展，但不同的节点向某一节点收集到的数据是不同的。

 版权所有 © 华为技术有限公司

图 10-11 AllToAll 和 AllGather 使用对比



应用于模型并行；模型并行里的矩阵转置；数据并行到模型并行的矩阵转置。

定义

```
struct AllToAllVParam {
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    std::vector<int64_t> sendCounts;
    std::vector<int64_t> sdispls;
    std::vector<int64_t> recvCounts;
    std::vector<int64_t> rdispls;
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rank	int	0	当前卡所属通信编号。
rankSize	int	0	通信的卡的数量。
rankRoot	int	0	主通信编号

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
sendCounts	std::vector< int64_t >	[]	表示发送数据量的数组。 例如：若发送的数据类型为float32，sendCounts[i] = n 表示本rank发给rank i n个float32数据。
sdispls	std::vector< int64_t >	[]	表示发送偏移量的数组。 sdispls[i] = n表示本rank从相对于输入起始位置的偏移量为n的位置开始发送数据给rank i。
recvCounts	std::vector< int64_t >	[]	表示接收数据量的数组。 例如：若发送的数据类型为float32，recvCounts[i] = n 表示本rank从rank i收到n个float32数据。
rdispls	std::vector< int64_t >	[]	表示接收偏移量的数组。 rdispls[i] = n表示本rank从相对于输出起始位置的偏移量为n的位置开始接收rank i的数据。
backend	std::string	"hccl"	通信计算类型，仅支持"hccl"。
hcclComm	HcclComm	nullptr	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	通信模式，**CommMode**类型枚举值。hccl多线程只支持外部传入通信域方式。
rankTableFile	std::string	-	集群信息的配置文件路径，适用单机以及多机通信场景，当前仅支持hccl后端场景。 若单机配置了ranktable，则以ranktable来初始化通信域。 配置请参见《TensorFlow 1.15模型迁移指南》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。
commDomain	std::string	-	通信device组用通信域名标识，多通信域时使用，该算子无需传此参数。
rsv[64]	uint8_t	{0}	预留参数。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	"hccl"：float16/float/int8/int16/int32/int64/bf16	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
output	[1,recvCountsSum] recvCountsSum为参数recvCounts的所有元素之和	"hccl": float16/float/int8/int16/int32/int64/bf16 数据类型和输入相同	ND	输出tensor，最后一维的shape为参数recvCounts的所有元素之和。

规格约束

- 数组参数sendCounts，recvCounts，sdispls，rdispls的长度等于ranksize，且数组中元素值都大于0。
- 当前只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 各个通信卡的输入tensor维度相等。
- sendCounts，recvCounts数组元素之和不能溢出int64，且recvCounts的数组和要大于0。对于rdispls中的任一元素，recvCounts[i] + rdispls[i] 不能大于recvCountsSum（即output的最后一维），对于sdispls中的任一元素，sendCounts[i] + sdispls[i] 不能大于输入tensor的数据量。如输入tensor shape为[3, 4, 5]，数据量为3*4*5 = 60，在进行AllToAllV时会被视为shape为[60]的tensor进行计算。
- rank、rankSize、rankRoot需满足以下条件。
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：

```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```

 版权所有 © 华为技术有限公司

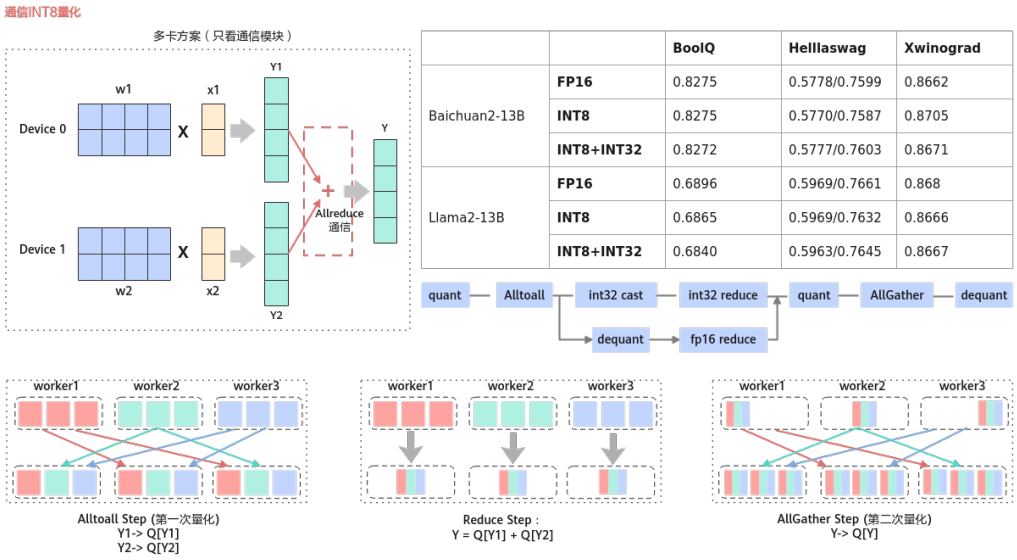
10.13 AllToAllVV2Operation

功能

向通信域内所有通信卡发送数据（数据量可以通过参数定制），并从所有通信卡接收数据（数据量可以通过参数定制）。

算子上下文

图 10-12 通信 INT8 量化中 AllToAll 的位置



		BoolQ	Helllaswag	Xwinograd
Baichuan2-13B	**FP16**	0.8275	0.5778/0.7599	0.8662
	INT8	0.8275	0.5770/0.7587	0.8705
	INT8+INT32	0.8272	0.5777/0.7603	0.8671
Llama2-13B	**FP16**	0.6896	0.5969/0.7661	0.868
	INT8	0.6865	0.5969/0.7632	0.8666
	INT8+INT32	0.6840	0.5963/0.7645	0.8667

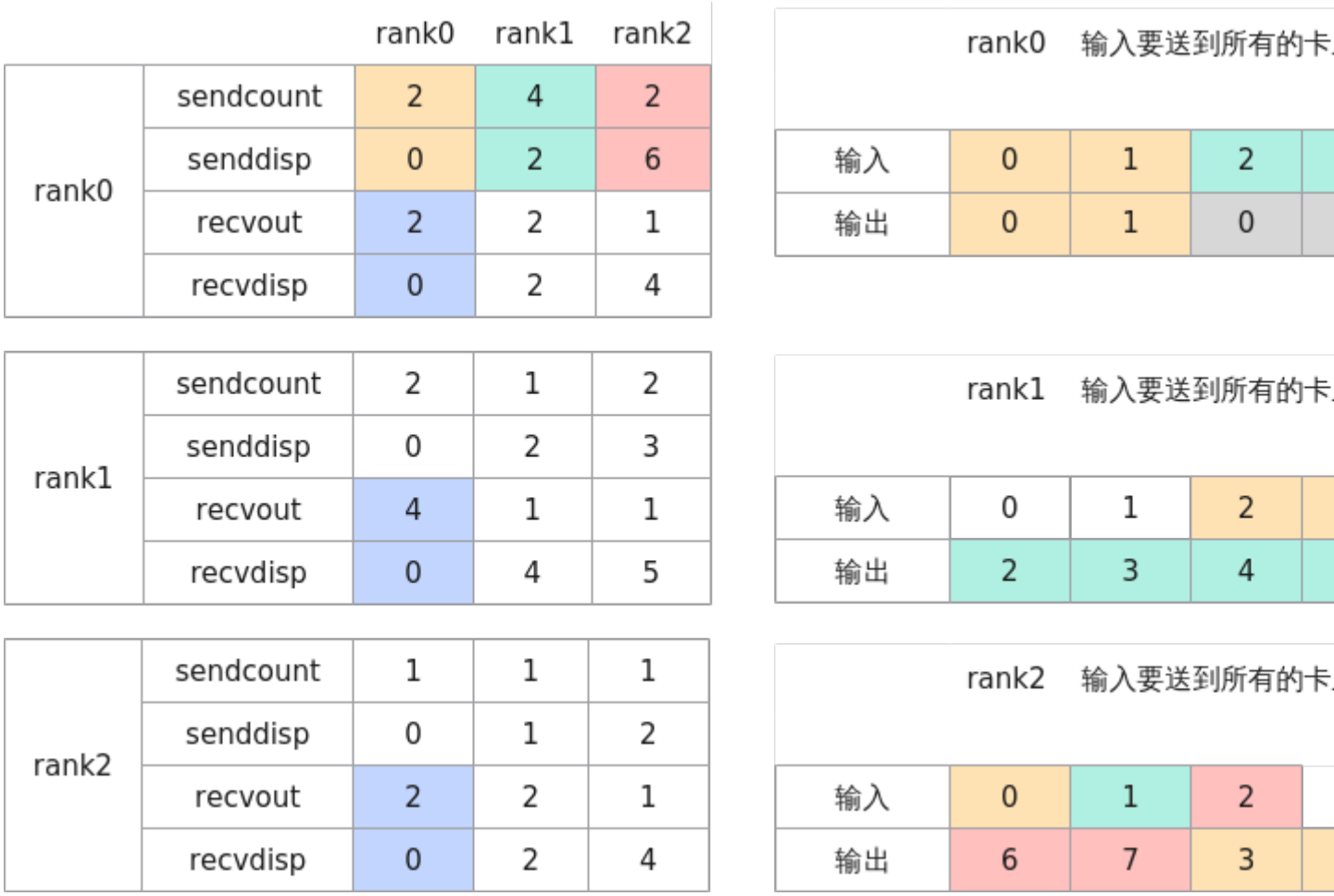
模型传输一个int8的量化输入tensor数据X,首先使用AlltoAll进行通信发送到各个节点上（这里使用int8进行通信，提升了通信速度）

然后使用reduce的sum操作对x进行求和，将int8数据反量化为float16, 最后使用AllGather进行通信将计算结果传输到各个节点上。（需求图中AllGather前后的量化，有可能会有精度损失，当前未实现。）

 版权所有 © 华为技术有限公司

算子功能实现描述

图 10-13 算子输入输出示例



计算过程示例（python）

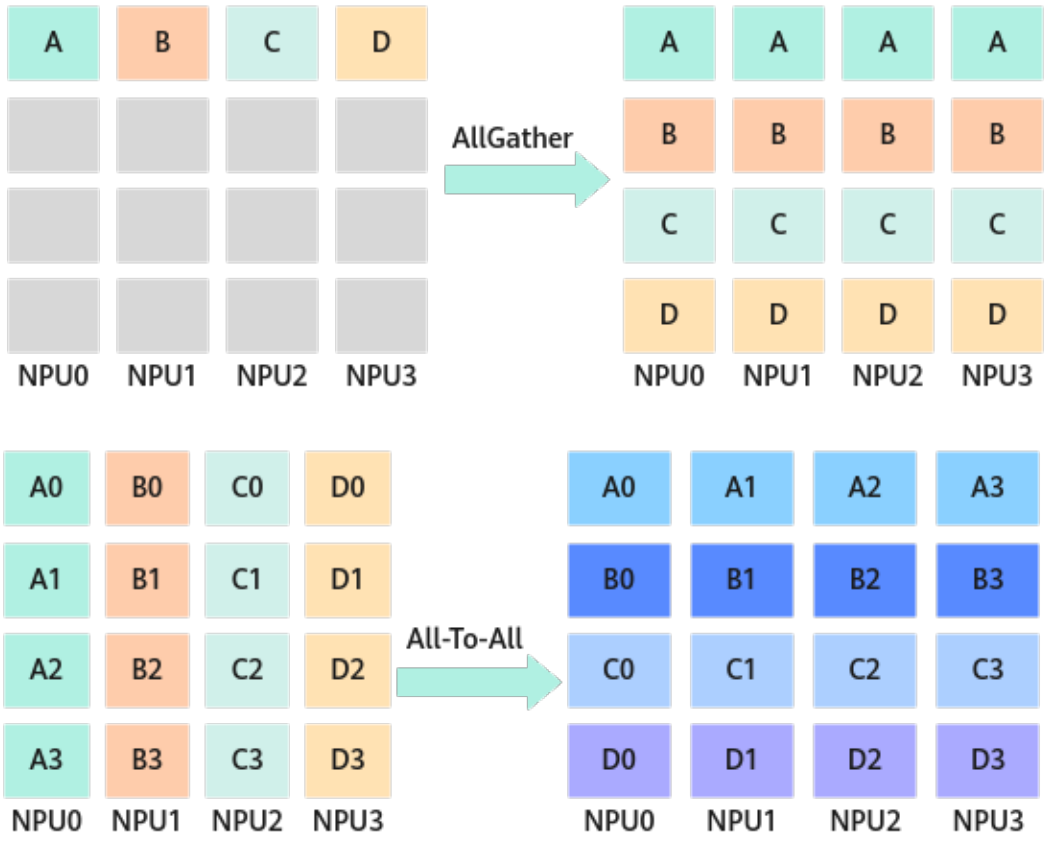
```
# 计算goldTensor
gold_outtensor = []
for j in range(len(recvout[rank])):
    gold_outtensor.append(tensorafters[j][senddisp[j][rank]:sendcount[j][rank] + senddisp[j][rank]])
gold_outtensor = [i for arr in gold_outtensor for i in arr]
```

使用场景

用于将数据发送到各个节点上，多对多。AllToAll是对Allgather的扩展，相比于Allgather，AllToAll不同的节点从某一节点收集到的数据是不同的。

 版权所有 © 华为技术有限公司

图 10-14 AllToAll 和 AllGather 使用对比



应用于模型并行；模型并行里的矩阵转置；数据并行到模型并行的矩阵转置。

使用示例：

```
>>> rank0 input
tensor([[0,1,2,3],
        [4,5,6,7]], device='npu:0')  shape[2,4]
>>> rank0 sendcount
tensor([2, 4], device='npu:0')  shape[2]
>>> rank0 sdispls
tensor([0, 2])
>>> rank0 recvCounts
tensor([2, 2])
>>> rank0 rdispls
tensor([0, 2])
>>> rank0 tensorForInferShape
tensor([3, 3, 3, 3])

>>> rank1 input
tensor([[0, 1, 2],
        [3, 4, 6]], device='npu:1')  shape[2,3]
>>> rank1 sendcount
tensor([2, 1], device='npu:1')  shape[2]
>>> rank1 sdispls
tensor([0, 2])
>>> rank1 recvCounts
tensor([4, 1])
>>> rank1 rdispls
tensor([0, 4])
>>> rank1 tensorForInferShape
tensor([3, 3, 3, 3, 3])

>>> rank0 output
```

 版权所有 © 华为技术有限公司

```
tensor([[0, 1, 0, 1]], device='npu:0')  shape[1,4]
>>> rank1 output
tensor([[2, 3, 4, 5, 2]], device='npu:1')  shape[1,5]
```

定义

```
struct AllToAllVV2Param {
    int rank = -1;
    int rankSize = 0;
    int rankRoot = 0;
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rank	int32	-1	当前卡所属通信编号。
rankSize	int32	0	通信的卡的数量，不能为0。
rankRoot	int32	0	主通信编号。
backend	std::string	"hccl"	通信计算类型，仅支持"hccl"。
hcclComm	HcclComm	nullptr	hccl通信域接口获取的地址指针。默认为空，加速库为用户创建;若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。

commMode	CommMode	COMM_MULTI_PROCESS	通信模式，CommMode类型枚举值。hccl多线程只支持外部传入通信域方式。
			enum CommMode : int {
			COMM_UNDEFINED = -1, //!< 未定义
			COMM_MULTI_PROCESS, //!< 指定多进程通信
			COMM_MULTI_THREAD, //!< 指定多线程通信
			};

rankTableFile	std::string	-	集群信息的配置文件路径，用于多机通信场景，当前仅支持hccl后端场景。 集群信息的配置文件路径，适用单机以及多机通信场景，当前仅支持hccl后端场景,若单机配置了rankTable，则以ranktable来初始化通信域。 配置请参见《 TensorFlow 1.15模型迁移指南 》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
commDomain	std::string	-	通信device组用通信域名标识，多通信域时使用，当前仅支持hccl。
rsv[64]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	是否必选	描述
x	[dim_0, dim_1, ..., dim_n]	"hccl": float16/int8	ND	是	输入tensor。
sendCount	1[rankSize]	int64	ND	否	表示发送数据量的数组，为host侧tensor。 例如，若发送的数据类型为float16，sendCounts[i] = n 表示本rank发给rank_i n个float16数据。
sdispls	1[rankSize]	int64	ND	否	表示发送偏移量的数组，为host侧tensor。sdispls[i] = n表示本rank从相对于输入起始位置的偏移量为n的位置开始发送数据给rank_i。
recvCounts	1[rankSize]	int64	ND	否	表示接收数据量的数组，为host侧tensor。例如，若发送的数据类型为float16，recvCounts[i] = n 表示本rank从rank_i收到n个float16数据。
rdispls	1[rankSize]	int64	ND	否	表示接收偏移量的数组，为host侧tensor。rdispls[i] = n表示本rank从相对于输入起始位置的偏移量为n的位置开始接收rank_i的数据。
tensorForInferShape	[recvCountsSum]	int8	ND	否	shape为recvCounts的所有元素之和，用于infer shape。

 版权所有 © 华为技术有限公司

输出

参数	维度	数据类型	格式	是否必选	描述
output	[1,recvCountsSum]	"hccl": float16/int8	ND	否	输出tensor，最后一维的shape为参数recvCounts的所有元素之和。数据类型和输入相同。

规格约束

- 多卡场景要指定rank。
- sendCounts、recvCounts、sdispls、rdispls均可看作长度为ranksize的一维数组，且数组中元素值都大于等于0。
- 仅支持Atlas 推理系列产品和Atlas 800I A2 推理产品。
- sendCounts、recvCounts数组元素之和不能溢出int64，对于rdispls中的任一元素，recvCounts[i] + rdispls[i] 不能大于recvCountsSum（即output的最后一维），对于sdispls中的任一元素，sendCounts[i] + sdispls[i]不能大于输入tensor的数据量。如输入tensor shape为[3, 4, 5]，数据量为3*4*5 = 60，在进行AllToAllV时会被视为shape为[60]的tensor进行计算。
- rank、rankSize、rankRoot需满足以下条件。
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：

```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```

接口调用示例（Python）

```
def main_worker(rank, world_size,inTensorDtypes, sizes, random_seed):
    # init process group
    torch_npu.npu.set_device(int(rank))
    print(f'========== Process {rank} started, using device npu:{rank}.==========')
    # init all_to_allvv2_operation
    op_name = "AllToAllVV2Operation"
    all_to_allvv2_operation = torch.classes.OperationTorch.OperationTorch(
        "AllToAllVV2Operation")
    torch.manual_seed(random_seed)
    low = -100
    high = 100
    for inTensorDtype in inTensorDtypes:
        inTensors=[torch.tensor([[0,1,2,3],[4,5,6,7]],dtype=inTensorDtype),torch.tensor([[0,1,2],
[3,4,6]],dtype=inTensorDtype)]

GoldenTensors=[torch.tensor([[0,1,0,1]],dtype=inTensorDtype),torch.tensor([2,3,4,5,2],dtype=inTensorDtype)
]
        acl_param = json.dumps({"rank": rank, "rankSize": world_size,"rankRoot": 0, "backend": "hccl"})
        run_param = json.dumps({"sendCounts": sendcount[rank], "sdispls": senddisp[rank], "recvCounts":
```

 版权所有 © 华为技术有限公司

```
recvout[rank],
                            "rdispls": recvdis[rank]})
        host_list = [sendcount[rank], senddisp[rank], recvout[rank], recvdis[rank]]
        host_tensors = [np.array(tensor) for tensor in host_list]
        host_tensors = [torch.from_numpy(tensor).to(torch.int64) for tensor in host_tensors]
        host_tensors = [tensor.npu() for tensor in host_tensors]
        all_to_allvv2_operation.set_param(acl_param)
        all_to_allvv2_operation.set_varaintpack_param(run_param)
        print(f"host_tensors={host_tensors}")
        print(f"inTensors[rank].npu()={inTensors[rank].npu()}")
        print(f"host_tensors[0]={host_tensors[0]}")
        print(f"host_tensors[1]={host_tensors[1]}")
        print(f"host_tensors[2]={host_tensors[2]}")
        print(f"host_tensors[3]={host_tensors[3]}")
        atb_input5 = compute_out_tensor_dim1(recvout[rank])
        print(f"atb_input5 = {atb_input5}")
        result1 = all_to_allvv2_operation.execute(
            [inTensors[rank].npu(), host_tensors[0], host_tensors[1], host_tensors[2], host_tensors[3],
atb_input5])
        print(f"\n ==========\nresult1={result1}\n ==========\n")
        acl_out_tensor = result1[0]
        print(f"\n ==========\nacl_out_tensor={acl_out_tensor}\n ==========\n")
        torch.npu.synchronize()
        # assert result
        assert golden_compare(acl_out_tensor.cpu(), GoldenTensors[rank])
def test_all_to_allvv2_operation(self):
    world_size = 2
    random_seed = 123
    sizes = [[3, 4]]
    set_start_method('spawn', force=True)
    process_list = []
    inTensorDtypes = [torch.int8, torch.int16, torch.int32, torch.int64, torch.float32, torch.float16,
                      torch.bfloat16]
    for i in range(world_size):
        p = Process(target=main_worker,args=(i,world_size, inTensorDtypes, sizes, random_seed))
        p.start()
        process_list.append(p)
    for i in process_list:
        p.join()
```

10.14 AsStridedOperation

功能

根据指定参数，使用输入tensor生成一个数据重新排布过的输出tensor。

算子上下文

图 10-15 AsStridedOperation



 版权所有 © 华为技术有限公司

算子功能实现描述

AsStrided算子的功能是基于已有的张量创建一个新的视图（view），可以通过指定形状（shape）和步长（strides）来重新排列元素。

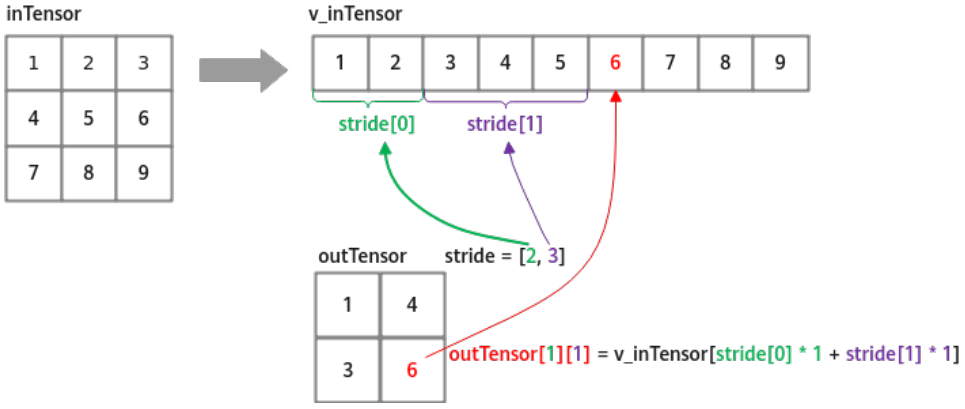
映射关系（以输入为 3x3 shape、输出为 2x2 shape 为例）

- 输入
  ```
  inTensor = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
  opParam.size = [2, 2]
  opParam.stride = [2, 3]
  opParam.offset = [0]
  ```
- 输出
  ```
  outTensor = [[1, 4], [3, 6]]
  ```

映射关系为：outTensor[i][j] -> v_inTensor[i * stride[0] + j * stride[1] + offset[0]]。

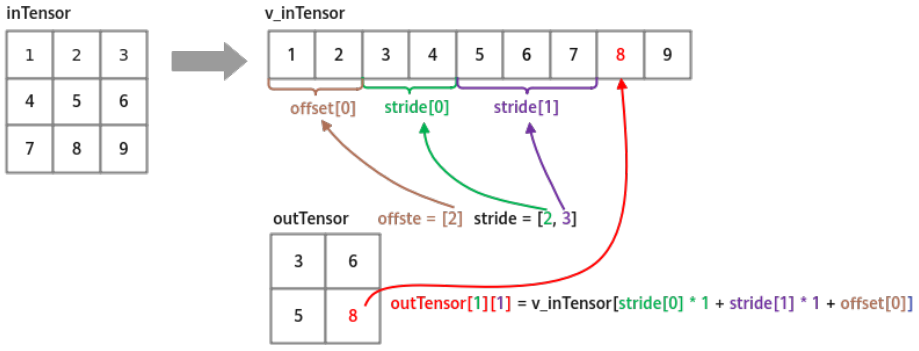
其中：v_inTensor 为 输入tensor展开的一维向量。

图 10-16 AsStrided 算子实现原理（offset 为 0 时）



- offset不为0时的输出：
  若加上 offset：opParam.offset = [2]。
  则计算后输出为：outTensor = [[3, 6], [5, 8]]。

  图 10-17 AsStrided 算子实现原理（offset 不为 0 时）



定义

```
struct AsStridedParam {
    SVector<int64_t> size;
```

 版权所有 © 华为技术有限公司

```
    SVector<int64_t> stride;
    SVector<int64_t> offset;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
size	SVector<int64_t>	-	输出tensor的shape。 “size”的长度要求小于或等于8且各元素要求大于0。
stride	SVector<int64_t>	-	用于从输入tensor推导输出tensor的各维度的步长。 “stride”的长度要求与size一致，各元素要求大于或等于0。
offset	SVector<int64_t>	-	输出tensor内存相对于输入tensor内存的偏移，作为常数使用。 “offset”的长度要求为1且元素要求大于或等于0。
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[din_0, ..., din_n]	float16/int64	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
y	[dout_0, ..., dout_m]	float16/int64	ND	输出tensor。

规格约束

Param参数约束：

- offset.size() == 1，offset.at(i) ≥ 0。
- size.size() ≤ 8，size.at(i) > 0。
- stride.size() == size.size()，stride.at(i) ≥ 0。

 版权所有 © 华为技术有限公司

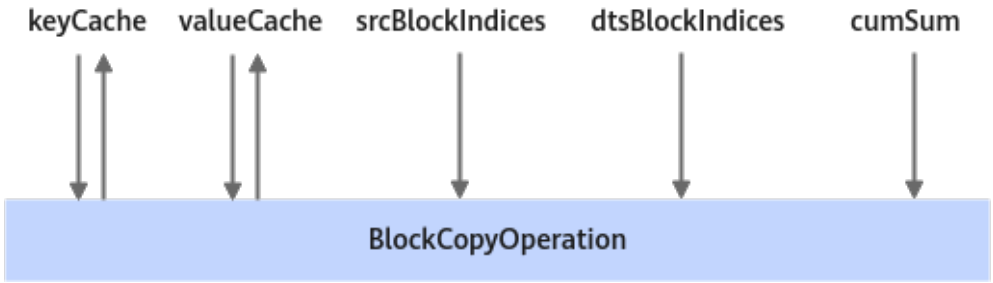
10.15 BlockCopyOperation

功能

通过指定源block index和目的block index，将keyCache和valueCache中的数据从一系列源block拷贝到一系列目的block（ keyCache和valueCache做相同的拷贝处理 ）。

算子上下文

图 10-18 BlockCopyOperation



定义

```
struct BlockCopyParam {
    uint8_t rsv[16] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rsv[16]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
keyCache	[num_blocks, block_size, num_head, head_size]	float16/ bf16/int8	ND	key矩阵。
valueCache	[num_blocks, block_size, num_head, head_size]	float16/ bf16/int8	ND	value矩阵。 数据类型和格式应该与keyCache保持一致。
srcBlockIndices	[src_count]，最长为[num_blocks]	int32	ND	keyCache、valueCache源位置block索引，值域范围为[0,num_blocks)。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
dstBlockIndices	[dst_count]，最长为[num_blocks]	int32	ND	keyCache、valueCache目标位置block索引，dstBlockIndices[cumSum[i-1]:cumSum[i]-1]为srcBlockIndices[i]对应的目标位置List（特殊的，dstBlockIndices[0:cumSum[0]-1]为srcBlockIndices[0]对应的目标位置List），值域范围为[0,num_blocks)。
cumSum	[src_count]	int32	ND	cumSum[i]为源位置srcBlockIndices[i]对应的目标位置List在dstBlockIndices中的结束位置。

输出

参数	维度	数据类型	格式	描述
keyCacheNew	[num_blocks, block_size, num_head, head_size]	float16/bf16/int8	ND	block拷贝后的key矩阵，作为输出。输出keyCacheNew与输入keyCache指向同一地址，即进行原地修改。 数据类型和格式应该与keyCache保持一致。
valueCacheNew	[num_blocks, block_size, num_head, head_size]	float16/bf16/int8	ND	block拷贝后的value矩阵，作为输出。输出valueCacheNew与输入valueCache指向同一地址，即进行原地修改。 数据类型和格式应该与valueCache保持一致。

规格约束

- 当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 一个block index不能同时出现在srcBlockIndices和dstBlockIndices中。
- srcBlockIndices中的block index不重复，如有重复则应合并为一项，其对应的dstBlockIndices取并集。
- dstBlockIndices中的block index不重复，即不能有两个不同的block拷贝到同一个block中。
- 每个srcBlockIndex至少有一个dstBlockIndex与之对应（i > 0时，cumSum[i] - cumSum[i - 1] ≥ 1；cumSum[0] ≥ 1）。

 版权所有 © 华为技术有限公司

- 根据前述4条规则，srcBlockIndices和dstBlockIndices的size应满足：dst_count ≥ src_count，src_count + dst_count ≤ num_blocks。

10.16 BroadcastOperation

功能

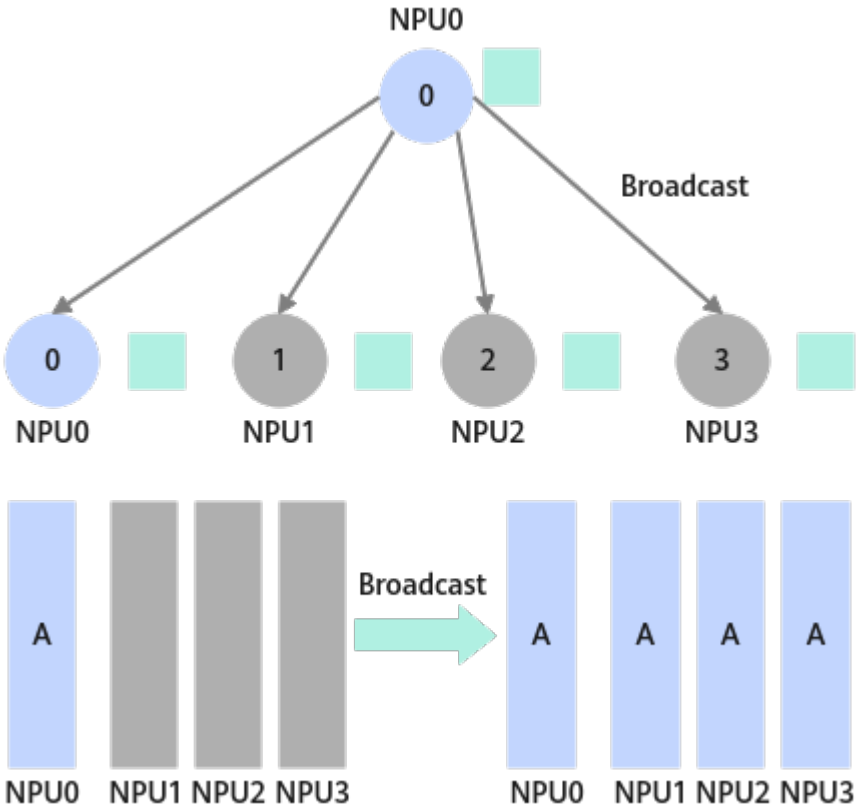
将通信主卡上的数据广播到其他每张卡上。该算子为原地操作，输出tensor与输入tensor内存地址相同。

该算子涉及多卡相关操作，可根据实际需求配置HCCL相关环境变量，具体请参见《CANN 环境变量参考》中的“集合通信”章节。

使用场景

1对多，广播方式。主节点0将数据发送到其他节点，且数据内容相同。

图 10-19 通信原理示意图



使用示例

参数设置

```
rankRoot=0
rankSize=2
```

输入输出

 版权所有 © 华为技术有限公司

```
npu0（通信主卡）：
x=[67,60,-75]
output=[67,60,-75]
npu1：
x=[0,0,0]
output=[67,60,-75]
```

定义

```
struct BroadcastParam {
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string backend = "hccl";
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rank	int	0	当前卡所属通信编号。
rankSize	int	0	通信的卡的数量。
rankRoot	int	0	主通信编号。
hcclComm	HcclComm	nullptr	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	通信模式，**CommMode**类型枚举值。hccl多线程只支持外部传入通信域方式。
backend	string	“hccl”	通信后端指示，支持“hccl”和“lccl”。
rankTableFile	string	-	集群信息的配置文件路径，用于单机以及多机通信场景，当前仅支持HCCL后端场景。 若单机配置了ranktable，则以ranktable来初始化通信域。 配置请参见《TensorFlow 1.15模型迁移指南》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。
commDomain	string	-	通信device组用通信域名标识，多通信域时使用。当backend为lccl时，commMode为多进程时，commDomain需要设置0-63。
rsv[64]	uint8_t	{0}	预留参数。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	• "hccl": float16/float/int8/int16/int32/int64/bf16 • "lccl": float16/float/int8/int16/int32/int64/bf16	ND	输入tensor。

规格约束

该算子仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

rank、rankSize、rankRoot需满足以下条件。

- 0 ≤ rank < rankSize
- 0 ≤ rankRoot < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：
```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```
- 当前不支持单device上跑多个HCCP进程实例，不能单卡同时跑多个通信算子。比如运行通信算子场景下，不支持同一张卡上跑多个模型。

10.17 CohereLayerNormOperation

功能

一种归一化方法，可以将网络层输入根据-1维归一化到[0, 1]之间。

定义

```
struct CohereLayerNormParam {
    float epsilon = 1e-5;
    uint8_t rsv[32] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
epsilon	float	1e-5	大于0	否	epsilon，归一化时加在分母上防止除0。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
rsv	uint8_t	{0}	-	否	预留字段。

Command R Plus 模型归一化功能

- **功能描述**

  针对Command R Plus模型，对多batch数据用于表示根据最后一维进行归一化操作。

- **计算公式**

$$y = \frac{x - E[x[, -1]]}{\sqrt{Var[x[, -1]] + \varepsilon}} * \gamma$$

  E[, -1]和Var[, -1]分别为最后一维的均值和方差，ε 为epsilon，γ 为权重。

- **硬件支持情况**

硬件型号	支持情况
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持float16、bf16

- **输入**

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	• 最后一维的大小要32字节对齐。 • 维度只能为3或4。
gamma	[dim_n-1, dim_n]	float16/bf16	ND	• 最后一维的大小要32字节对齐。 • 维度只能为2。 • x和gamma最后两维的维度数一致。

- **输出**

参数	维度	数据类型	格式	描述
result	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	x、result和gamma的数据格式一致。

 版权所有 © 华为技术有限公司

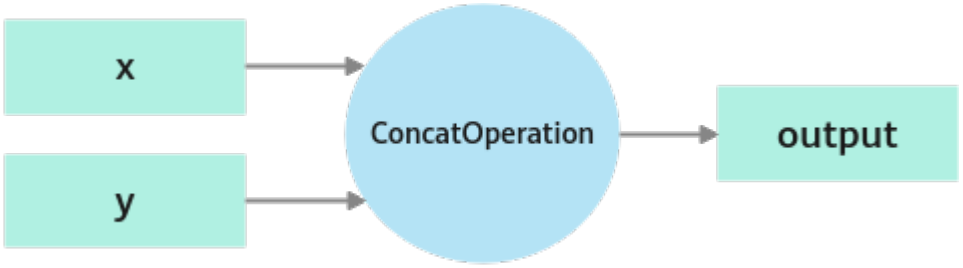
10.18 ConcatOperation

功能

将两个输入张量在指定维度拼接成一个输出张量。

算子上下文

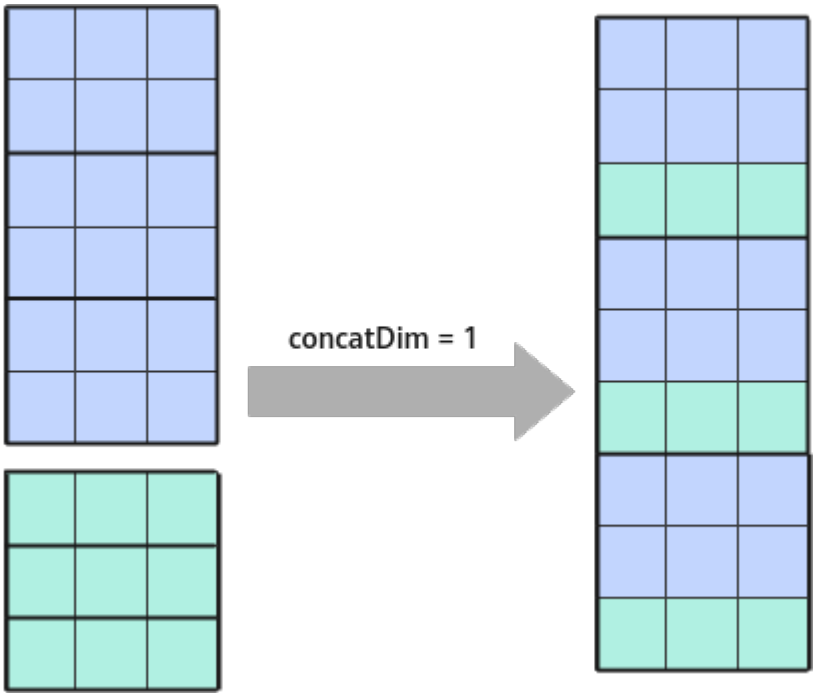
图 10-20 ConcatOperation



算子功能实现

Concat 算子的功能是将两个输入张量在指定维度拼接成一个输出张量。

图 10-21 Concat 算子实现原理



定义

```
struct ConcatParam {
    int concatDim = 0;
    uint8_t rsv[12] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	描述
concatDim	int	0	指定拼接的维度索引。 假设拼接的维度索引为k，x维度数为dimNum： ● 当concatDim为负数时，其含义是从最高维度开始访问，k=dimNum+concatDim。例如concatDim = -1，则拼接维度k为dimNum - 1。 ● 否则k=concatDim。
rsv[12]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，... ，x_dim_n]	float16/bf16	ND	需要被拼接的tensor。
y	[y_dim_0，y_dim_1，... ，y_dim_n]	float16/bf16	ND	需要被拼接的tensor。除了y_dim_k之外，其余维度与x相同。

输出

参数	维度	数据类型	格式	描述
output	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	拼接后的tensor。

规格约束

- **Param参数约束**
  - dimNum为被拼接tensor的维数。
  - -dimNum ≤ concatDim ≤ dimNum - 1。
- **输入约束**
  - 输入x和y的维数相等。
  - 输入x和y的维度大小，除了cancatDim维之外，其他维度要求相同。

接口调用示例

- 输入

```
x shape = [3, 2, 3]:
[[[1.0, 1.0, 1.0],
  [1.0, 1.0, 1.0]],
```

 版权所有 © 华为技术有限公司

```
 [[1.0, 1.0, 1.0],
  [1.0, 1.0, 1.0]],
 [[1.0, 1.0, 1.0],
  [1.0, 1.0, 1.0]]]
y shape = [3, 1, 3]:
[[[2.0, 2.0, 2.0]],
 [[2.0, 2.0, 2.0]],
 [[2.0, 2.0, 2.0]]]
param.concatDim = 1（或-2）
```

- 输出

```
output shape = [3, 3, 3]
[[[1.0, 1.0, 1.0],
  [1.0, 1.0, 1.0],
  [2.0, 2.0, 2.0]],
 [[1.0, 1.0, 1.0],
  [1.0, 1.0, 1.0],
  [2.0, 2.0, 2.0]],
 [[1.0, 1.0, 1.0],
  [1.0, 1.0, 1.0],
  [2.0, 2.0, 2.0]]]
```

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include <random>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"
#include "demo_util.h"
int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);
    // 配置Op参数
    atb::infer::ConcatParam opParam;
    opParam.concatDim = 1;  // 设定拼接轴为1
    // 准备VariantPack
    atb::VariantPack variantPack;
    std::vector<int64_t> inputXShape = {1, 3, 2};
    std::vector<int64_t> inputYShape = {1, 2, 2};
    std::vector<int64_t> outputShape = {1, 5, 2};
    std::vector<float> inTensorXData = {0, 1, 2, 3, 4, 5};
    std::vector<float> inTensorYData = {6, 7, 8, 9};
    std::vector<float> outputRefData = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    atb::Tensor inTensorX =
        CreateTensorFromVector(context, stream, inTensorXData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
inputXShape);
    atb::Tensor inTensorY =
        CreateTensorFromVector(context, stream, inTensorYData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
inputYShape);
    atb::Tensor outTensor = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, outputShape);
    variantPack.inTensors = {inTensorX, inTensorY};
    variantPack.outTensors = {outTensor};
    // 申请ConcatOp
```

 版权所有 © 华为技术有限公司

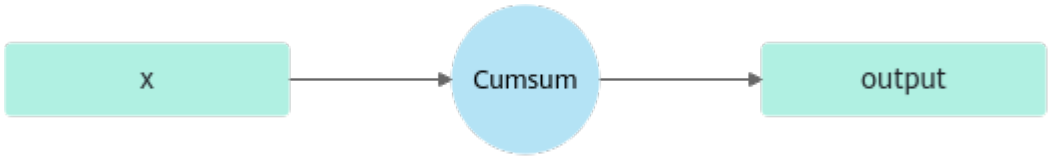
```
    atb::Operation *concatOp = {nullptr};
    CHECK_STATUS(atb::CreateOperation(opParam, &concatOp));
    uint64_t workspaceSize = 0;
    // ATB Operation 第一阶段接口调用：对输入输出进行检查，并根据需要计算workspace大小
    CHECK_STATUS(concatOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // ATB Operation 第二阶段接口调用：执行算子
    CHECK_STATUS(concatOp->Execute(variantPack, workspacePtr, workspaceSize, context));
    CHECK_STATUS(aclrtSynchronizeStream(stream)); // 流同步，等待device侧任务计算完成
    // 资源释放
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(concatOp));
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));
    CHECK_STATUS(aclFinalize());
    std::cout << "Concat demo success!" << std::endl;
    return 0;
}
```

10.19 CumsumOperation

功能

返回给定axes上的累计和。

图 10-22 CumsumOperation



功能实现

输入：

- **数组**：cumsum 的输入可以是多维数组。
- **指定轴**：可以选择沿着哪个轴（维度）进行累积和操作。默认情况下，对于一维数组，累积和是从第一个元素开始向后计算；对于多维数组，需要指定沿哪个轴进行累积。

输出：

- **累积和的结果**：输出的形状与输入相同，每个元素是该元素以及之前所有元素的和。

 版权所有 © 华为技术有限公司

计算公式

如果输入是大小为N的一维容器，那么输出也为大小为N的一维容器，其中元素的值为：

$$y_i = x_1 + x_2 + x_3 + \ldots + x_i$$

定义

```
struct CumsumParam {
    SVector<int64_t> axes;
    bool exclusive = false;
    bool reverse =  false;
    uint8_t rsv[14] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
axes	SVector<int64_t>	-	指定axis轴(维度)上计算累加和，只能包含一个轴索引。 “axes”的值必须小于输入x的维度数。
exclusive	bool	false	在某一个轴上的累加结果从第几个元素开始，默认为false。 ● true：从第一个元素开始。（暂不支持） ● false：从第0个元素开始。
reverse	bool	false	正向累加或逆向累加，默认为false。 ● true：输出逆向累加。（暂不支持） ● false：输出正向累加。
rsv[14]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/ bf16	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
y	[dim_0, dim_1, ..., dim_n]	float16/ bf16	ND	输出tensor。

 版权所有 © 华为技术有限公司

规格约束

- 参数"axes"的值必须小于输入x的维度数。
- 参数"axes"数组大小必须为1。
- 参数"reverse"必须为false。
- 参数"exclusive"必须为false。

10.20 DynamicNTKOperation

功能

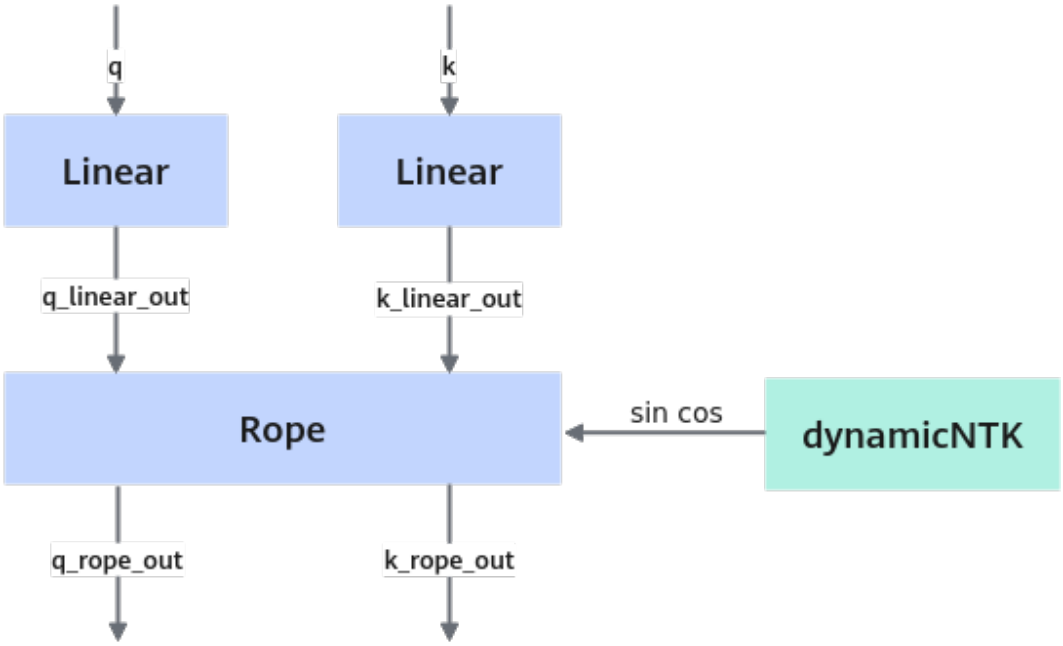
推理的长度大于训练序列长度时，生成ROPE算子使用的旋转矩阵（即下面公式中的sin/cos矩阵）。

推理长度小于等于训练序列长度时，不进行插值；推理长度大于训练长度时，放大base动态插值。

将输入的token序列的位置信息positionIds和invFreq进行外积，再cos/sin运算得到最终的Rotary Embedding的结果。

算子上下文

图 10-23 DynamicNTK 算子在模型中的位置



dynamicNTK算子输出sin，cos。

q，k，sin，cos一起输入到Rope算子中输出旋转后的q，k：

 版权所有 © 华为技术有限公司

$$f\left(\begin{matrix} q, m \end{matrix}\right) = \begin{pmatrix} cos\, m\theta_0 & -sin\, m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ sin\, m\theta_0 & cos\, m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & cos\, m\theta_1 & -sin\, m\theta_1 & \cdots & 0 & 0 \\ 0 & 0 & sin\, m\theta_1 & cos\, m\theta_1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & cos\, m\theta_{d/2-1} & -sin\, m\theta_{d/2-1} \\ 0 & 0 & 0 & 0 & \cdots & sin\, m\theta_{d/2-1} & cos\, m\theta_{d/2-1} \end{pmatrix}$$

算子功能分析

DynamicNTK用于推理长度大于训练长度时，对embedding做特殊处理，根据seqLen长度，对base参数进行缩放，最终生成新的cos、sin。由于Llama、Qwen、Yarn等模型对于cos、sin的生成方案存在一部分差异，DynamicNTK算子实现不同模型下公共的处理逻辑，差异化处理逻辑由模型实现。

- **算子计算公式**

  将输入的token序列的位置信息positionIds和invFreq进行外积，再cos/sin运算得到最终的Rotary Embedding的结果。

- **具体计算过程：**

  先通过每个batch的seqLen取出每个batch的positionIds，invFreq

  对每一个batch：

```
freq = torch.einsum(‘i,j->ij’, positionIds, invFreq)
emb = torch.cat((freq, freq), dim=-1)
sinOut = emb.sin()
cosOut = emb.cos()
```

定义

```
struct DynamicNTKParam {
    aclDataType outDataType = ACL_DT_UNDEFINED;
    uint8_t rsv[12] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
outDataType	aclDataType	ACL_DT_UNDEFINED	选择输出数据类型的参数。 • ACL_DT_UNDEFINED：默认值，此默认类型不可用，用户需选择其他两项输出数据类型。 • ACL_FLOAT16：设置输出的dtype为float16。 • ACL_BF16：设置输出的dtype为bf16。
rsv[12]	uint8_t	{0}	预留参数。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
positionIds	[ntokens]	int32	ND	多个batch的token位置序列，格式如下：[0,1,2,…,batch1Len-1,0,1,2…batch2Len-1,0,1,2…]。
InvFreqs	[batch, headSize / 2]	float	ND	每个batch的位置逆频。
seqLens	[batch]	int32	ND	每个batch的序列长度。

参数InvFreqs的计算公式为：

$$inv_freqs\left[batch\right] = \frac{1.0}{base[batch]^{troch.arange(0,dim,2)/dim}}$$

输出

参数	维度	数据类型	格式	描述
sin	[ntokens, headSize]	float16/bf16	ND	输出的Rotary Embedding的sin矩阵。
cos	[ntokens, headSize]	float16/bf16	ND	输出的cos矩阵，数据类型及shape与输出sin保持一致。

约束

- headSize（即输出tensor的最后一维）小于等于2048，并且是32的倍数。
- batch小于等于16。
- nTokens（即输入positionIds的维度）小于等于256000。
- InvFreqIn数组数据范围：[0,1)，seqLens数组中数据大于0，且数组和为nTokens。

10.21 ElewiseOperation

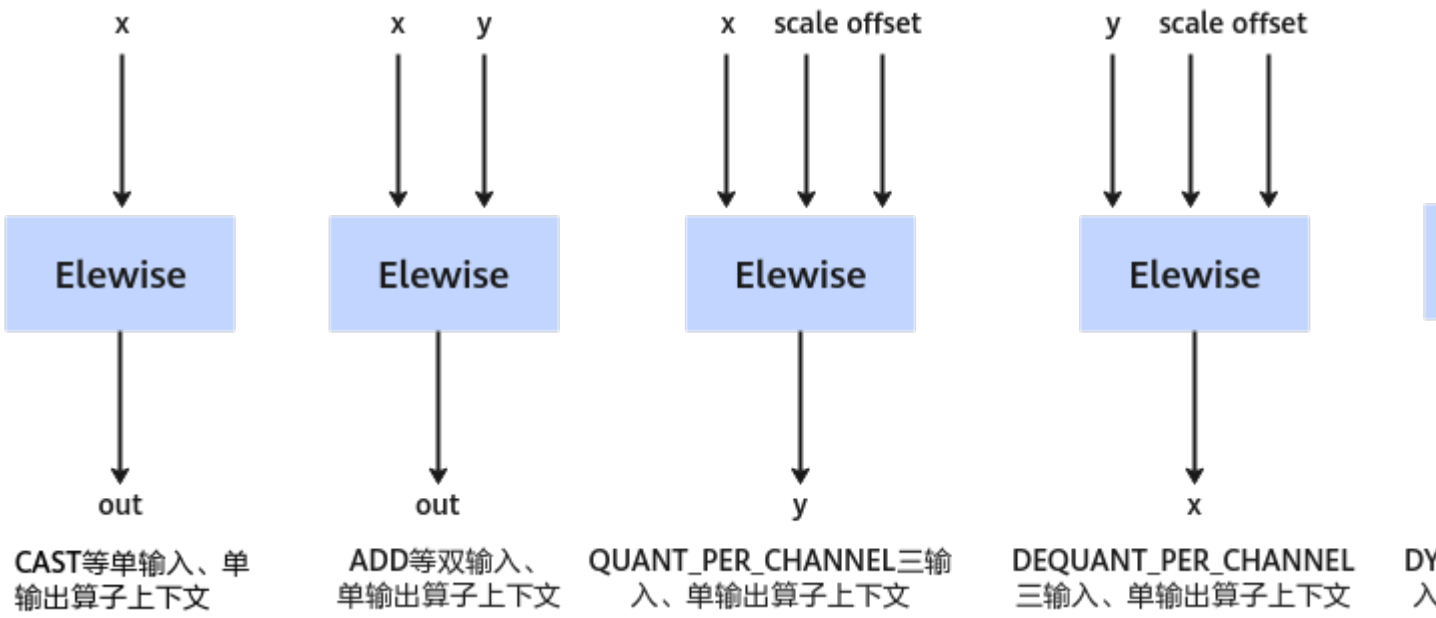
10.21.1 功能描述

常用的逐元素数值计算集合。

 版权所有 © 华为技术有限公司

10.21.2 算子上下文

图 10-24 ElewiseOperation 算子上下文



10.21.3 硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 200I/500 A2 推理产品	支持	ELEWISE_CAST、ELEWISE_MUL、ELEWISE_MULS、ELEWISE_ADD，仅支持上述运算且只支持float16数据类型。
Atlas 推理系列产品	支持	-
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

10.21.4 定义

```
struct ElewiseParam {
    enum ElewiseType: int {
        ELEWISE_UNDEFINED = 0,
        ELEWISE_CAST,
        ELEWISE_MULS,
        ELEWISE_COS,
        ELEWISE_SIN,
        ELEWISE_NEG,
        ELEWISE_QUANT,
        ELEWISE_LOGICAL_NOT,
        ELEWISE_ADD,
        ELEWISE_MUL,
        ELEWISE_REALDIV,
        ELEWISE_LOGICAL_AND,
```

 版权所有 © 华为技术有限公司

```
        ELEWISE_LOGICAL_OR,
        ELEWISE_LESS,
        ELEWISE_GREATER,
        ELEWISE_SUB,
        ELEWISE_EQUAL,
        ELEWISE_QUANT_PER_CHANNEL,
        ELEWISE_DEQUANT_PER_CHANNEL,
        ELEWISE_DYNAMIC_QUANT,
        ELEWISE_TANH,
        ELEWISE_TYPE_MAX
    };

    struct QuantParam {
        float inputScale = 1.0f;
        bool asymmetric = false;
        int inputOffset = 0;
        uint8_t rsv[20] = {0};
    };

    struct MulsParam {
        float varAttr = 0.0f;
        uint8_t rsv[12] = {0};
    };

    ElewiseType elewiseType = ELEWISE_UNDEFINED;
    QuantParam quantParam;
    MulsParam mulsParam;
    aclDataType outTensorType = ACL_DT_UNDEFINED;
    uint8_t rsv[8] = {0};
};
```

 版权所有 © 华为技术有限公司

10.21.5 参数列表

成员名称	类型	默认值	描述
elewiseType	ElewiseType	ELEWISE_UNDEFINED	计算类型。 • ELEWISE_UNDEFINED：默认值，未定义，此默认类型不可用，用户需配置此项参数。 • ELEWISE_CAST：数据类型转换。 • ELEWISE_MULS：向量逐元素乘值。 • ELEWISE_COS：逐元素计算余弦值。 • ELEWISE_SIN：逐元素计算正弦值。 • ELEWISE_NEG：逐元素取相反数。 • ELEWISE_QUANT：量化。 • ELEWISE_LOGICAL_NOT：逐元素逻辑非。 • ELEWISE_ADD：逐元素相加。 • ELEWISE_MUL：向量与向量逐元素相乘。 • ELEWISE_REALDIV：向量与向量逐元素相除。 • ELEWISE_LOGICAL_AND：逐元素逻辑与。 • ELEWISE_LOGICAL_OR：逐元素逻辑或。 • ELEWISE_LESS：逐元素判断是否小于。 • ELEWISE_GREATER：逐元素判断是否大于。 • ELEWISE_SUB：逐元素相减。 • ELEWISE_EQUAL：逐元素判断是否相等。 • ELEWISE_QUANT_PER_CHANNEL：每个通道量化。 • ELEWISE_DEQUANT_PER_CHANNEL：每个通道反量化。 • ELEWISE_DYNAMIC_QUANT：逐行动态量化。 • ELEWISE_TANH：逐元素计算双曲正切值。 • ELEWISE_TYPE_MAX：边界值，仅用于判断是否出界，所有情况不能取该值。
quantParam	struct	-	量化参数，具体请参见**表10-1**。
mulsParam	struct	-	乘值参数，具体请参见**表10-2**。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
outTensorType	aclDataType	ACL_DT_UNDEFINED	指定数据类型转换输出的数据类型。
rsv[8]	uint8_t	{0}	预留参数。

表 10-1 QuantParam 成员

成员名称	类型	默认值	描述
inputScale	float	1.0f	量化的步长。
asymmetric	bool	false	动态量化的是否为非对称量化。 ● false：symmetric，对称量化。 ● true：asymmetric，非对称量化。
inputOffset	int	0	量化的偏移度。
rsv[20]	uint8_t	{0}	预留参数。

表 10-2 MulsParam 成员

成员名称	类型	默认值	描述
varAttr	float	0.0f	向量乘的值。
rsv[12]	uint8_t	{0}	预留参数。

10.21.6 输入输出

ELEWISE_CAST（转化数据类型）

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float16/float/int64/int32/bf16	ND	被转换类型的输入。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
out	[dim_0，dim_1，...，dim_n]	float16/float/int64/int32/bf16	ND	被转换类型的输出。 • 在Atlas 800I A2 推理产品/Atlas A2 训练系列产品上，支持float16和float、int32和int64、bf16和float数据类型互相转化。 • 在Atlas 推理系列产品上，支持float16和float数据类型互相转化。

ELEWISE_MULS（向量逐元素乘值）

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float16/float/bf16	ND	输入。
out	[dim_0，dim_1，...，dim_n]	float/float16/bf16	ND	输出。数据类型与x保持一致。

ELEWISE_COS（逐元素计算余弦值）

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float/float16/bf16	ND	输入。
out	[dim_0，dim_1，...，dim_n]	float/float16/bf16	ND	输出。数据类型与x保持一致。

ELEWISE_SIN（逐元素计算正弦值）

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float/float16/bf16	ND	输入。
out	[dim_0，dim_1，...，dim_n]	float/float16/bf16	ND	输出。数据类型与x保持一致。

 版权所有 © 华为技术有限公司

ELEWISE_NEG（逐元素取相反数）

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float16/bf16	ND	输入。
out	[dim_0，dim_1，...，dim_n]	float16/bf16	ND	输出。数据类型与x保持一致。

ELEWISE_QUANT（量化）

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float16	ND	输入。
out	[dim_0，dim_1，...，dim_n]	int8	ND	输出。
x/out不支持Atlas 推理系列产品。				

ELEWISE_LOGICAL_NOT（逐元素逻辑非）

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	int8	ND	输入。
out	[dim_0，dim_1，...，dim_n]	int8	ND	输出。

ELEWISE_ADD（逐元素相加）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	float/float16/bf16/int32/int64	ND	输入1。
y	[y_dim_0，...，y_dim_n]	与x一致	ND	输入2。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	与x一致	ND	输出。输出、输入1、输入2数据类型需保证一致。

ELEWISE_MUL（向量与向量逐元素相乘）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	float16/bf16	ND	输入1。
y	[y_dim_0，...，y_dim_n]	与x一致	ND	输入2。
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	与x一致	ND	输出。输出、输入1和输入2数据类型需保证一致

ELEWISE_REALDIV（向量与向量逐元素相除）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	float/float16/bf16	ND	输入1。
y	[y_dim_0，...，y_dim_n]	与x一致	ND	输入2。
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	与x一致	ND	输出。输出、输入1和输入2数据类型需保证一致。

 版权所有 © 华为技术有限公司

ELEWISE_LOGICAL_AND（逐元素逻辑与）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int8	ND	输入1。
y	[y_dim_0，...，y_dim_n]	int8	ND	输入2。
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int8	ND	输出。

ELEWISE_LOGICAL_OR（逐元素逻辑或）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int8	ND	输入1。
y	[y_dim_0，...，y_dim_n]	int8	ND	输入2。输入1和输入2的元素值只能为0或1。
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int8	ND	输出。

ELEWISE_LESS（逐元素判断是否小于）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int64/float/float16/bf16	ND	输入1。
y	[y_dim_0，...，y_dim_n]	与x一致	ND	输入2。输入1、输入2数据类型需保证一致。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int8	ND	输出。

ELEWISE_GREATER（逐元素判断是否大于）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int64/float/float16/bf16	ND	输入1。
y	[y_dim_0，...，y_dim_n]	与x一致	ND	输入2。输入1、输入2数据类型需保证一致。
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int8	ND	输出。

ELEWISE_SUB（逐元素相减）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	float16/int64/bf16	ND	输入1。
y	[y_dim_0，...，y_dim_n]	与x一致	ND	输入2。输入1、输入2数据类型需保证一致。
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	与x一致	ND	输出。

 版权所有 © 华为技术有限公司

ELEWISE_EQUAL（逐元素判断是否相等）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	float/float16/bf16	ND	输入1。
y	[y_dim_0，...，y_dim_n]	与x一致	ND	输入2。输入1、输入2数据类型需保证一致。
out	[x_dim_0，x_dim_1，...，x_dim_n，y_dim_0，...，y_dim_n]	int8	ND	输出。

ELEWISE_QUANT_PER_CHANNEL（每个通道量化）

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，...，x_dim_n，dim_0，...，dim_n]	float16/bf16	ND	输入1。
scale	[dim_0，...，dim_n]	float16/bf16	ND	输入2。scale中元素要求不为0。可以为标量。输入1，输入2数据类型需保证一致。
offset	[dim_0，...，dim_n]	int8/float	ND	输入3。可选，offset可以为空Tensor，可以为标量。
y	[x_dim_0，x_dim_1，...，x_dim_n，dim_0，...，dim_n]	int8	ND	量化输出结果。

ELEWISE_DEQUANT_PER_CHANNEL（每个通道反量化）

参数	维度	数据类型	格式	描述
y	[x_dim_0，x_dim_1，...，x_dim_n，dim_0，...，dim_n]	int8	ND	输入1。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
scale	[dim_0，...，dim_n]	float16	ND	输入2。可以为标量。
offset	[dim_0，...，dim_n]	int8	ND	输入3。可选，offset可以为空Tensor，可以为标量。输入1，输入3数据类型需保证一致。
x	[x_dim_0， x_dim_1，...， x_dim_n， dim_0，...，dim_n]	float16	ND	反量化输出结果。

ELEWISE_DYNAMIC_QUANT（逐行动态量化）

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float16/bf16	ND	输入向量，n>1。
z	[dim_0，dim_1，...，dim_n]	int8	ND	量化输出结果。
scale	[dim_0，...，dim_n-1]	float	ND	输出，反量化所需的scale。
offset	[dim_0，...，dim_n-1]	float	ND	输出，反量化所需的offset，在“asymmetric”为“true”时才返回。当前不支持asymmetric为true的场景。

ELEWISE_TANH（逐元素计算双曲正切值）

参数	维度	数据类型	格式	描述
x	[dim_0，...，dim_n]	float16	ND	输入。
out	[dim_0，...，dim_n]	float16	ND	输出。

10.21.7 规格约束

- ELEWISE_ADD、ELEWISE_MUL、ELEWISE_REALDIV、ELEWISE_SUB、ELEWISE_LOGICAL_AND、ELEWISE_LOGICAL_OR、ELEWISE_LESS、ELEWISE_GREATER、ELEWISE_EQUAL计算类型将会对输入进行广播后再进行指定操作。

 版权所有 © 华为技术有限公司

- 输入x、y对应维度的对应值要求相同或至少其中一个为1。
- ElewiseDeQuantPerChannel 只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 在Atlas 推理系列产品上，ELEWISE_QUANT_PER_CHANNEL只支持32位对齐的数据。
- 在ELEWISE_DYNAMIC_QUANT计算类型输入x中，有如下约束
  - 在Atlas 800I A2 推理产品/Atlas A2 训练系列产品上：float16类型尾轴可支持的最大数据量为26624；bf16类型尾轴可支持的最大数据量为7552；暂时只支持对称量化。
  - 在Atlas 推理系列产品上：尾轴只支持32位对齐的数据，尾轴可支持的最大数据量为4096，暂时只支持对称量化。

10.21.8 算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

using namespace atb;
using namespace std;

const int VECTOR_SIZE = 4;      // 向量的大小
const float INIT_VALUE = 2.0f;  // 向量的初始值

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。序列长度向量，等于1时，为增量或全量；大于1时，为全量
 * @param tokenOffsetHost host侧tensor。计算完成后的token偏移
 * @param layerId layerId，取cache的kv中哪一个kv进行计算
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    uint32_t dim0 = 2;
    uint32_t dim1 = 2;
    // 创建tensor0
    std::vector<float> tensormul0(VECTOR_SIZE, INIT_VALUE);
    atb::Tensor tensorMul0 =
        CreateTensorFromVector(contextPtr, stream, tensormul0, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{dim0, dim1});
    // 创建tensor1
    std::vector<float> tensormul1(VECTOR_SIZE, INIT_VALUE);
    atb::Tensor tensorMul1 =
        CreateTensorFromVector(contextPtr, stream, tensormul1, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{dim0, dim1});
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {tensorMul0, tensorMul1};
    return inTensors;
}
```

 版权所有 © 华为技术有限公司

```
/**
 * @brief 创建一个ELEWISE_MUL的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::ElewiseParam mulParam;
    mulParam.elewiseType = atb::infer::ElewiseParam::ElewiseType::ELEWISE_MUL;
    atb::Operation *op = nullptr;
    CHECK_STATUS(atb::CreateOperation(mulParam, &op));
    return op;
}

int main(int argc, char **argv)
{
    // 1.设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // ELEWISE_MUL示例
    atb::Operation *op = PrepareOperation();
    // 准备输入张量
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);                          // 放入输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {2, 2});  // 创建输出
tensor
    variantPack.outTensors.push_back(tensorOut);                                       // 放入输出tensor

    // setup阶段，计算workspace大小
    uint64_t workspaceSize = 0;
    CHECK_STATUS(op->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }

    // execute阶段
    op->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放内存
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(op));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "elewise_mul demo success!" << std::endl;
    return 0;
}
```

 版权所有 © 华为技术有限公司

10.22 FaUpdateOperation

功能

将各SP域PA算子的输出的中间结果lse、attention out两个局部结果更新成全局结果。

硬件支持情况

硬件型号	支持情况
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持

定义

```
struct FaUpdateParam {
    enum FaUpdateType {
        DECODE_UPDATE = 0,
    };
    FaUpdateType faUpdateType = DECODE_UPDATE;
    uint32_t sp = 1;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
faUpdateType	FaUpdateType	DECODE_UPDATE	DECODE_UPDATE	否	指定下标需要执行的操作类型，目前只支持取默认值。
sp	uint32_t	1	[1, 8]	否	序列并行的并行度SP。
rsv[64]	uint8_t	{0}	[0]	否	预留参数。

输入

参数	维度	数据类型	格式	描述
lse	[sp, batch * seqLen * headNum]	float	ND	各SP域计算的lse。
localout	[sp, batch * seqLen * headNum, head_size]	float	ND	各SP域计算的output。

 版权所有 © 华为技术有限公司

输出

参数	维度	数据类型	格式	描述
output	[batch * seqLen * headNum, head_size]	float	ND	全局的output。

规格约束

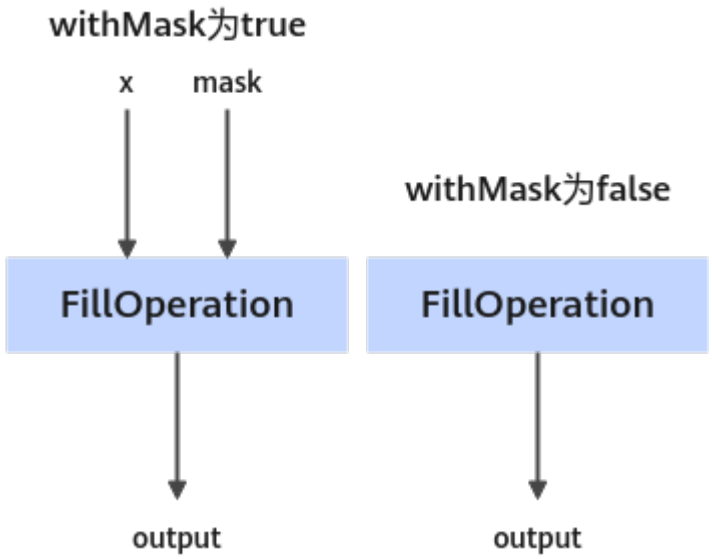
head_size取值范围为[8, 512]，且必须是8的倍数。

10.23 FillOperation

功能

将指定位置设置为value值或者生成一个指定Shape的Tensor并填充为value。

图 10-25 FillOperation



功能实现

Fill 算子的功能是将指定位置设置为 value 值或者生成一个指定 Shape 的 Tensor 并填充为 value。

- **MaskedFill 功能（withMask 为 true）：**
  若 x 里的某元素，其所对应 mask 里同位置元素值为 true 或 1，则 output 该位置元素被设置为 value 值，output 其余位置元素值与 x 相同。支持广播功能。

 版权所有 © 华为技术有限公司

图 10-26 Fill 算子实现原理

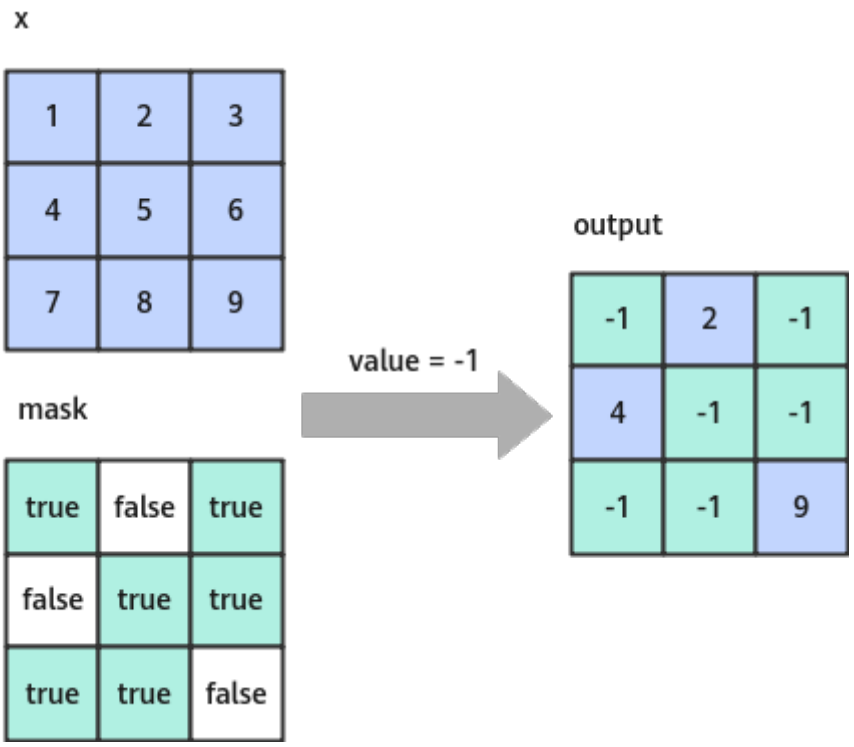
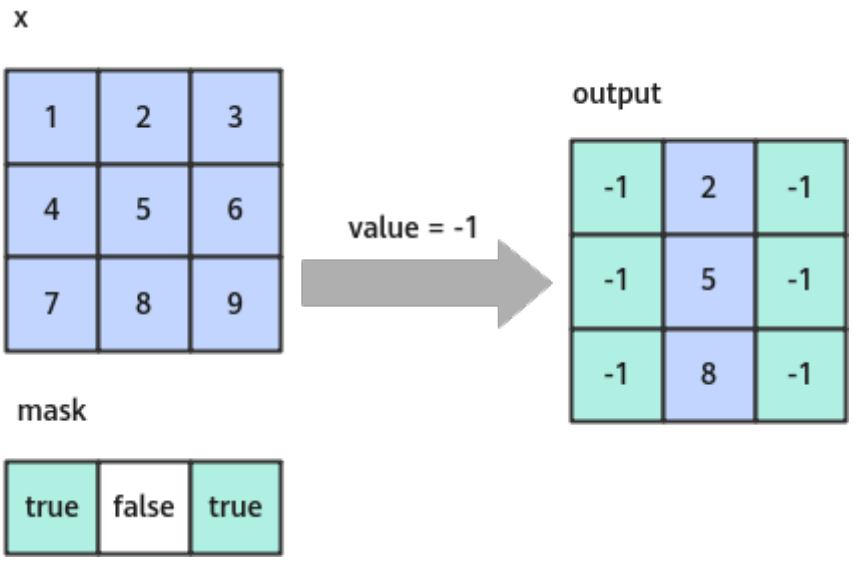


图 10-27 Fill 算子实现原理（广播场景）



- **Fill 功能（withMask 为 false）：**
  生成一个由 outDim 指定 shape 的 tensor，所有元素数值均为 value。

定义

```
struct FillParam {
    bool withMask = true;
    SVector<float> value;
    SVector<int64_t> outDim;
    uint8_t rsv[8] = {0};
}
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	描述
withMask	bool	true	是否Masked Fill。
value	SVector<float>	[]	填充的元素，value是一个只含有一个元素的SVector。
outDim	SVector<float>	[]	withMask = false时，表示输出tensor的Shape。
rsv[8]	uint8_t	{0}	预留参数。

- **withMask为true**
  将由mask指定的位置的元素设置为value值。
  输入tensors数量为2，不需要指定outDim。
- **withMask为false**
  生成一个由outDim指定Shape的tensor并将其每个元素填充为value。
  输入tensors数量为0，需指定outDim。

输入

参数	维度	数据类型	格式	描述
x	[dx_0, ..., dx_n]	float16/int32/bf16	ND	仅withMask为true时输入。
mask	[dmask_0, ..., dmask_m]	int8/bool	ND	仅withMask为true时输入，通过设置某些位置上的数值为 true 或 1，将x相同位置上的数据设置为value值。

输出

参数	维度	数据类型	格式	描述
output	• withMask为true时，和x相同 • withMask为false时，由outDim指定	• withMask为true时，和x相同 • withMask为false时，只能为float16	ND	输出tensor。

 版权所有 © 华为技术有限公司

规格约束

- **Param参数约束**

  a. value.size() == 1。

- **输入输出约束**
  - 输入x不可以被broadcast。
  - 输入mask的每个元素只能是0或者1。
  - withMask 为 true 时，mask shape 与 x 一致或可以 broadcast 为 x 的 shape。

接口调用示例

- **示例1：MaskedFill 非广播场景**
  - 输入

    ```
    param.withMask = true
    param.value = [-1]
    x = [[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]]
    mask = [[ true, false,  true],
           [false,  true,  true],
           [ true,  true, false]]
    ```

  - 输出

    ```
    output = [[-1,  2, -1],
             [ 4, -1, -1],
             [-1, -1,  9]]
    ```

- **示例2：MaskedFill 广播场景**
  - 输入

    ```
    param.withMask = true
    param.value = [-1]
    x = [[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]]
    mask = [true, false, true]
    ```

  - 输出

    ```
    output = [[-1,  2, -1],
             [-1,  5, -1],
             [-1,  8, -1]]
    ```

- **示例3：Fill 场景**
  - 输入

    ```
    param.withMask = false
    param.value = [1]
    param.outDim = [2, 3]
    ```

  - 输出

    ```
    output = [[1, 1, 1],
             [1, 1, 1]]
    ```

10.24 FusedAddTopkDivOperation

功能

Sigmoid，Add，GroupTopk，Gather，ReduceSum，RealDiv，Muls算子的功能融合。

算子支持两种模式：常规模式（物理专家模式）、逻辑专家模式。

 版权所有 © 华为技术有限公司

首先，每个专家经过sigmoid激活函数、Add偏置后，得出[bs, expert_num]的专家评分。其次每组经过DeepSeekV3的组级TOPK，专家级TOPK选出k个专家。

在常规模式下，这些专家作为物理专家的ID，会直接输出，用于指导后续的通信算子将每个token发往k个专家对应的卡上。

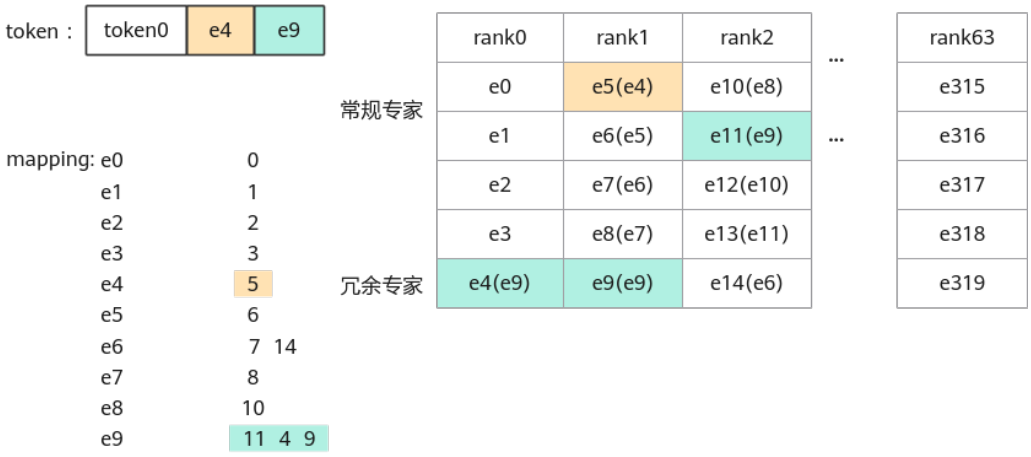
图 10-28 常规专家映射



在存在冗余专家场景下，每个物理专家会属于不同的卡，此时衍生出逻辑专家的概念，即每个卡上的不同专家为逻辑专家，一个物理专家映射到多个逻辑专家，逻辑专家ID/每个卡的专家数=对应卡的RankID，逻辑专家ID%每个卡的专家数=在卡上的位置。

在逻辑专家模式下，新增mappingNum指示每个物理专家被映射到的逻辑专家数，新增mappingTable指示每个物理专家被映射到的逻辑专家ID，算子内使用伪随机逻辑，随机选择其中一个逻辑专家进行映射，并更正输出indices为逻辑专家ID。

图 10-29 逻辑专家映射



硬件支持情况

硬件型号	支持情况
Atlas 200I/500 A2 推理产品	不支持
Atlas 推理系列产品	不支持
Atlas 训练系列产品	不支持

 版权所有 © 华为技术有限公司

硬件型号	支持情况
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持

定义

```
struct FusedAddTopkDivParam {
    uint32_t groupNum = 1;
    uint32_t groupTopk = 1;
    uint32_t n = 1;
    uint32_t k = 0;
    ActivationType activationType = ACTIVATION_SIGMOID;
    bool isNorm = true;
    float scale = 1.0f;
    bool enableExpertMapping = false;
    uint8_t rsv[27] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
groupNum	uint32_t	1	>0	是	分组数量。
groupTopk	uint32_t	1	>0	是	选择k个组。
n	uint32_t	1	>0	是	组内选取n个最大值求和。
k	uint32_t	1	>0	是	topk选取前k个值。
activationType	ActivationType	ACTIVATION_SIGMOID	ACTIVATION_SIGMOID	是	激活类型。
isNorm	bool	true	true/false	是	是否归一化。
scale	float	1.0	-	是	归一化后的乘系数。
enableExpertMapping	bool	false	true/false	是	是否使能物理专家向逻辑专家的映射。false时输入2个tensor，true时输入4个tensor。
rsv[27]	uint8_t	{0}	所有值均为0	否	预留字段，用于保证版本间接口兼容性。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
x	[a, b]	float16/float32/bf16	ND	输入tensor。
addNum	[b]	float16/float32/bf16	ND	输入tensor，用于与x相加。数据类型和格式与x一致。
mappingNum	[b]	int32	ND	enableExpertMapping为false时不启用，true时输入tensor，每个物理专家被实际映射到的逻辑专家数量。
mappingTable	[b, c] c<=128	int32	ND	enableExpertMapping为false时不启用，true时输入tensor，物理专家/逻辑专家映射表。

输出

参数	维度	数据类型	格式	描述
y	[a, k]	float32	ND	输出tensor。
indices	[a, k]	int32	ND	输出tensor。

规格约束

- b为groupNum的整数倍。
- groupTopk <= groupNum。
- k <= b。
- b >= groupNum * n。
- b <= groupNum * 32。
- 若b >= 32，则groupNum = 8。
- max_redundant_expert_num代表最大可能出现的额冗余专家数，目前最大支持128。
- mapping_num里面的元素值： 0<=元素值<c。
- 含义：a=bs，b=expert_num，c= max_redundant_expert_num。

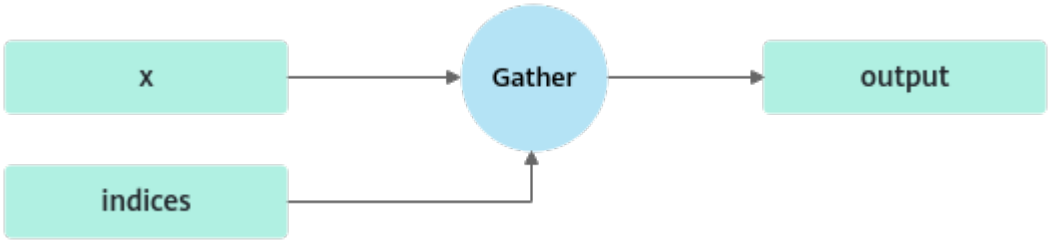
 版权所有 © 华为技术有限公司

10.25 GatherOperation

功能

从输入张量中根据索引收集切片，并将这些切片组合成一个新的张量。

图 10-30 GatherOperation 算子上下文



硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 200I/500 A2 推理产品	支持	输入x数据类型只支持float16。
Atlas 推理系列产品	支持	-
Atlas 800I A2 推理产品/ Atlas A2 训练系列产品	支持	-

定义

```
struct GatherParam {
    int64_t axis = 0;
    int64_t batchDims = 0;
    uint8_t rsv[16] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
axis	int64_t	0	≥0	是	指定要收集切片的轴。默认值为0。
batchDims	int64_t	0	≥0且≤axis	是	代表批处理的维度数。表示可以从每轮批处理的元素中分别取出满足要求的切片数据。 例如，如果batchDims=1，则代表在x的第（axis - batchDims）轴上有一个外循环，见**示例2**。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
rsv[16]	uint8_t	{0}	[0]	否	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float16/float/bf16/int32/uint32/int64	ND	输入tensor。
indexs	[dim_0，dim_1，...，dim_n]	int64/int32/uint32	ND	索引表，值必须在[0, x.shape[axis]]范围内，x与indexs的维数之和小于等于9。 indexs的维数必须大于等于“batchdims”。

输出

参数	维度	数据类型	格式	描述
output	[dim_0，dim_1，...，dim_n]	float16/float/bf16/int32/uint32/int64	ND	输出tensor。数据类型与x保持一致。

规格约束

- 索引表indices值必须在[0, x.shape[axis]]范围内，x与indexs的维数之和小于等于9。indexs的维数必须大于等于"batchdims"。
- 成员"axis" 必须大于等于0。
- "batchDims"必须大于或等于0，且小于或等于axis。

接口调用示例

- 示例1：

```
axis = 1;
batchDims = 0;
输入tensor为：
 x=  [[1,2,3],
      [4,5,6],
      [7,8,9]]
indices tensor为：
indices=[2,0]
```

 版权所有 © 华为技术有限公司

```
根据indices tensor的值，在axis轴获取x数据切片，output tensor为：
output=[[3, 1],
        [6, 4],
        [9, 7]]
```

- 示例2:

```
axis= 1;
batchDims = 1;
输入tensor为：
 x=  [[1,2,3],
      [4,5,6],
      [7,8,9]]
indices tensor为：
indices=[[1],
         [2],
         [0]]
因为batch_dims=1，则代表在第一个轴上（即轴0）进行批处理。在轴0上，将x[i]和indices[i]进行一一对应的gather处理，根据indices的值，在axis轴获取x的数据切片，其中i为batch轴的坐标，output tensor为：
output= [[2,],
         [6,],
         [7,]]
上面等价于：
def gather_fun(x, indices, axis):
       batch_dims=1
       res= []
    # 进行外循环
    for p,i in zip(x, indices):
        r = tf.gather(p, i, axis=axis-batch_dims)
        res.append(r)
    return tf.stack(res)
```

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。序列长度向量，等于1时，为增量或全量；大于1时，为全量
 * @param tokenOffsetHost host侧tensor。计算完成后的token偏移
 * @param layerId layerId，取cache的kv中哪一个kv进行计算
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    uint32_t dim0 = 3;
    uint32_t dim1 = 3;
    // 创建tensor0
    std::vector<float> tensorzero{1, 2, 3, 4, 5, 6, 7, 8, 9};
    atb::Tensor tensorZero =
        CreateTensorFromVector(contextPtr, stream, tensorzero, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{dim0, dim1});
    // 创建tensor1
    std::vector<int64_t> tensorone{2, 1};
    atb::Tensor tensorOne =
```

 版权所有 © 华为技术有限公司

```
        CreateTensorFromVector(contextPtr, stream, tensorone, ACL_INT64, aclFormat::ACL_FORMAT_ND, {2});
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {tensorZero, tensorOne};
    return inTensors;
}

/**
 * @brief 创建一个Gather的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::GatherParam gatherParam;
    gatherParam.axis = 0;
    gatherParam.batchDims = 0;
    atb::Operation *op = nullptr;
    CHECK_STATUS(atb::CreateOperation(gatherParam, &op));
    return op;
}

int main(int argc, char **argv)
{
    // 1.设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // Gather示例
    atb::Operation *op = PrepareOperation();
    // 准备输入张量
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);                          // 放入输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {2, 3}); // 创建输出
tensor
    variantPack.outTensors.push_back(tensorOut);                                       // 放入输出tensor

    // setup阶段，计算workspace大小
    uint64_t workspaceSize = 0;
    CHECK_STATUS(op->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }

    // execute阶段
    op->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream)); // 流同步，等待device侧任务计算完成

    // 释放内存
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(op)); // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context)); // context，全局资源，后释放
    std::cout << "Gather demo success!" << std::endl;
    return 0;
}
```

 版权所有 © 华为技术有限公司

10.26 GatherPreRmsNormOperation

功能

首先对ResIn进行Gather索引操作，然后与X相加，最后进行RmsNorm计算。

计算公式

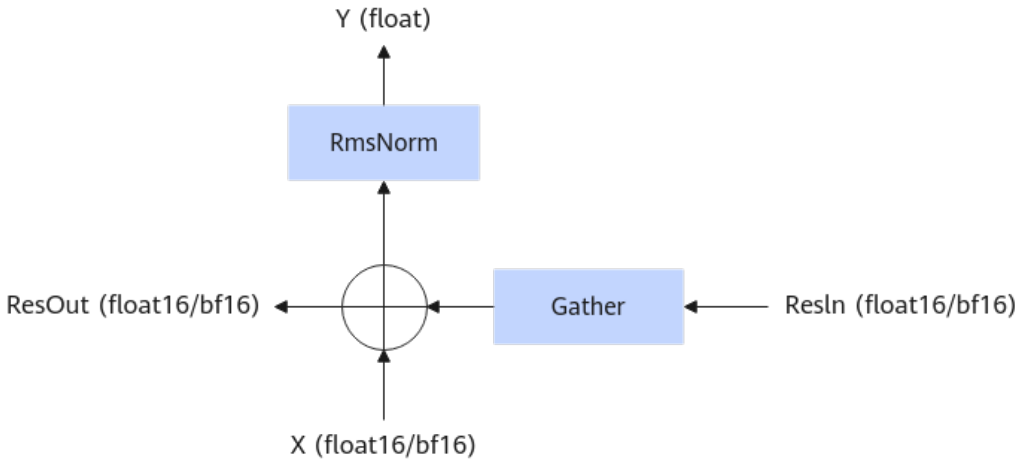
- Gather操作：根据indices中的值索引Resln中对应的行，得到ReslnAfterGather。
- Add操作：将X和RcslnAfterGather相加，得到RcsOut。
- Norm操作：

$$Y = RmsNorm(ResOut)$$

$$RmsNorm\left(x \right) = \frac{x}{RMS(x)} * Gamma$$

$$RMS\left(x \right) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2 + \varepsilon}$$

计算图



硬件支持情况

硬件型号	支持情况
Atlas 800I A2 推理产品	支持

 版权所有 © 华为技术有限公司

定义

```
struct GatherPreRmsNormParam {
    float epsilon = 1e-5;
    uint8_t rsv[28] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
epsilon	float	1e-5	(2e-38, 1e-5]	是	归一化时加在分母上防止除零。
rsv[28]	uint8_t	{0}	[0]	否	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_m, dim_n]	float16/bf16	ND	2维，末轴维度需要32字节对齐（16的倍数）且不大于7680。
resIn	[dim_res, dim_n]	float16/bf16	ND	2维，dim_m、dim_res可能不相等，与x的数据类型一样。
indices	[dim_m]	int32	ND	1维，元素取值范围为[0, dim_res)，其中dim_res为ResIn的第一个维度值。
gamma	[1, dim_n]或[dim_n]	float16/bf16	ND	1维或2维，与x的数据类型一样。

输出

参数	维度	数据类型	格式	描述
y	[dim_m, dim_n]	float	ND	数据类型、维度数和维度值与x一样。
ResOut	[dim_m, dim_n]	float16/bf16	ND	维度与X一致。

 版权所有 © 华为技术有限公司

10.27 GatingOperation

功能

主要功能为将token和专家的映射关系反转为专家与token的映射关系。算子输入为MoE模型每个token选中专家的索引，算子输出为MoE模型每个专家对应的token的索引。

该算子支持TP和EP场景，当参数“deviceExpert”为空时，为TP场景，否则为EP场景。

定义

```
struct GatingParam {
    int32_t topkExpertNum = 1;
    int32_t cumSumNum = 0;
    bool cumSumInt64 = false;
    std::vector<int32_t> deviceExpert;
    uint8_t rsv[16] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
topkExpertNum	int32_t	1	• 当cumSumNum为0时，取值为1 • 当cumSumNum不为0时，为(0, cumSumNum]。	是	每个token选中的专家数。
cumSumNum	int32_t	0	[0, 1024]	是	专家总数。
cumSumInt64	bool	false	true/false	否	输出的cumSum的类型是否为int64。 当为false时，输出的cumSum类型为int32。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
deviceExpert	std::vector< int32_t >	-	• 当cumSumNum为0时，列表必须为空。 • 当cumSumNum不为0时，列表中各个元素取值为[0, cumSumNum)。	否	当前device上的专家索引列表。 列表中各个元素取值范围为[0, cumSumNum)，且其中元素值不可重复。 当cumSumNum为0时，必须为空。
rsv[16]	uint8_t	{0}	[0]	否	预留参数。

输入

参数	维度	数据类型	格式	描述
topk	[tokenNum * topkExpertNum]	int32	ND	输入tensor。每个token选中的专家的index。值域为[0, cumSumNum - 1]；当cumSumNum为0时，值域为[0, 11300)。
idxArr	[tokenNum * topkExpertNum]	int32	ND	输入tensor。每个token原始的index，具体的值为[0,1,2,3,...]。
tokenNum表示token个数，tokenNum = batch * seqlen。				

输出

参数	维度	数据类型	格式	描述
tokenIndex	[tokenNum * topkExpertNum]	int32	ND	输出tensor。token重排以后原始的索引值。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
cumSum	[expertNum]	int32/ int64	ND	输出tensor。每个专家被选中的次数。当cumSumNum为0时，expertNum值为1；当deviceExpert不为空时，expertNum值为deviceExpert的元素个数，否则，expertNum值为cumSumNum，当cumSumInt64为True时，输出为int64类型。
originalIndex	[tokenNum * topkExpertNum]	int32	ND	输出tensor。token重排以后token的索引值。
validIndex	[1]	int32	ND	输出tensor。当deviceExpert不为空时输出。
tokenNum表示token个数，tokenNum = batch * seqlen。				

规格约束

非Atlas 800I A2 推理产品/Atlas A2 训练系列产品仅支持TP场景。

10.28 GroupedMatmulInplaceAddOperation

功能

算子Groupedmatmul和add融合，实现分组矩阵乘+InplaceAdd功能。

计算图

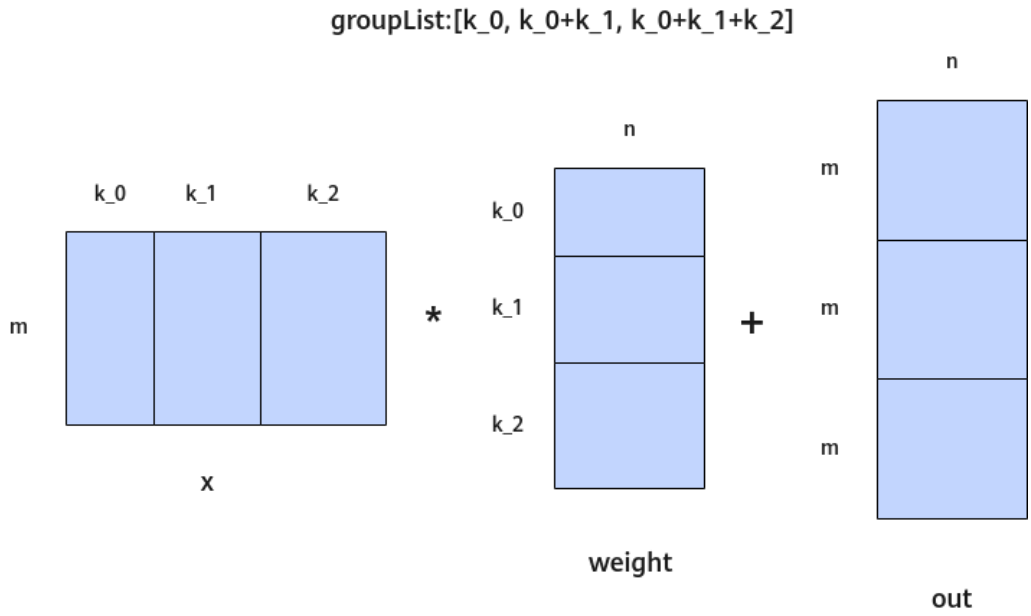
分组实现矩阵乘计算，并在输出地址原地累加。每组矩阵乘的维度中ki各不相同，x_i/weight_i可以在k_i上拼接。算子数学表达式为：

$$out_i[m_i, n_i] = x_i[mi, ki] * weight_i[k_i, n_i] + out_i[m_i, n_i], i=1,2...,g$$

其中g为分组个数，m_i/k_i/n_i为对应shape。

 版权所有 © 华为技术有限公司

图 10-31 计算示意图



定义

```
struct GroupedMatmulInplaceAddParam {
    bool transposeA = false;
    bool transposeB = false;
    uint8_t rsv[22] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	描述
transposeA	bool	False	false/true	代表矩阵A是否随路转置。
transposeB	bool	False	false	代表矩阵B是否随路转置，当前仅支持False。
rsv[22]	uint8_t	{0}	-	预留参数。

输入

参数	维度	数据类型	格式	是否必选	描述
x	[m, k]	float16/bf16	ND	是	矩阵乘的A矩阵，m取值范围（0，65535]。
weight	[k, n]	float16/bf16	ND	是	权重，矩阵乘的B矩阵，n取值范围（0，65535]。数据类型与x保持一致。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	是否必选	描述
groupList	[numGroup]	int64	ND	是	1维Tensor，代表矩阵乘分为多少个组，最大长度为128。
out	[numGroup * m, n]或[m, numGroup * n]	float	ND	是	既是输入tensor又是输出tensor，支持原地写，numGroup最大为128。

输出

参数	维度	数据类型	格式	是否必选	描述
out	[numGroup * m, n]或[m, numGroup * n]	float	ND	是	输出。

规格约束

- transposeB当前仅支持False，transposeA的取值以及矩阵A/B的维度需要满足矩阵乘的维度约束。
- 不支持Atlas 推理系列产品。
- 输出tensor与最后一个输入tensor为同一个tensor。
- groupList 为长度小于等于128的一维张量，代表A/B在k轴上分组的索引，groupList 中的后一个元素应大于等于前一个元素，最后一个元素大小等于k。

  例如：矩阵A维度为[5, 10]，矩阵B维度为[10, 4]，groupList维度为1维张量，numGroup为3，值分别为[3, 6, 10]，表示把k分为3组，分别为[0, 3)、[3, 6)、[6, 10)，分组进行矩阵乘。

10.29 GroupedMatmulWithRoutingOperation

功能

在MOE中，实现topK个专家权重与token激活值的乘法。

使用场景

UP：根据各个专家的权值，将token值分配给topK个专家。

DOWN：根据各个专家的权值，将topK个专家计算的结果重新整合。

 版权所有 © 华为技术有限公司

计算过程

- **UP**

  UP过程的计算公式如下：

  $$AcTensor_{Expertindex_e} * ExpertWeight_e, e \in [1, topK]$$

  其中，AcTensor为激活token值，Expertindex为专家-token索引，ExpertWeight为专家权重，计算过程如下：

  a. 根据当前专家的索引$Expertindex_e$，从AcTensor中获取当前专家的激活token值。
  b. 获取当前专家的权重，并与当前专家的激活token值做BatchMatmul。

- **DOWN**

  DOWN过程的计算公式如下：

  $$\sum_{e=1}^{topK} AcTensor_{Expertindex_e} * ExpertWeight_e, e \in [1, topK]$$

  其中，AcTensor为激活token值，Expertindex为专家-token索引，ExpertWeight为专家权重，计算过程如下：

  a. 根据当前专家的索$Expertindex_e$，从AcTensor中获取当前专家的激活token值。
  b. 获取当前专家的权重，并与当前专家的激活token值做BatchMatmul。
  c. 将当前token在所有专家上的结果相加。

定义

```
struct GroupedMatmulWithRoutingParam {
    enum GroupedMatmulType : int {
        GROUPED_MATMUL_UP = 0,
        GROUPED_MATMUL_DOWN
    };
    bool transposeB = true;
    int32_t topK = 0;
    GroupedMatmulType groupedMatmulType = GROUPED_MATMUL_UP;
    aclDataType outDataType = ACL_DT_UNDEFINED;
    uint8_t rsv[16] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
groupedMatmulType	GroupedMatmulType	GROUP_MATMUL_UP	• GROUP_MATMUL_UP • GROUP_MATMUL_DOWN	必选	GroupedMatmulType类型参数： • GROUP_MATMUL_UP：使用 UP 操作。 • GROUP_MATMUL_DOWN：使用 DOWN 操作。
transposeB	bool	true	false/true	非必选	是否转置 B 矩阵。
topK	int32_t	0	大于0	必选	选取 topK 个专家数量。
outDataType	aclDataType	ACL_DT_UNDEFINED	• ACL_DT_UNDEFINED • ACL_FLOAT16 • ACL_BF16	反量化场景必选	• 非量化场景时，数据类型必须与输入数据类型保持一致。 • 反量化场景时，必须设置输出数据类型。 • 可选输出类型：ACL_FLOAT16，ACL_BF16。
rsv[16]	uint8_t	{0}	[0]	-	预留参数。

输入输出

- **UP**

  非量化场景

  - **输入**

参数	维度	数据类型	格式	描述
AcTensor	[num_tokens, hidden_size_in]	float16/bf16	ND	激活值。
ExpertWeight	[num_experts, hidden_size_in, hidden_size_out]	float16/bf16	ND	专家权重。如果设置“transposeB”为true，则hidden_size_in和hidden_size_out位置调换。
ExpertCount	[num_experts]	int32	ND	专家 Token 数量。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
Expertindex	[num_tokens*topK]	int32	ND	专家 Token 索引。

- **输出**

参数	维度	数据类型	格式	描述
Result	[num_tokens*Topk, hidden_size_out]	float16/bf16	ND	输出。

量化场景

- **输入**

参数	维度	数据类型	格式	描述
AcTensor	[num_tokens, hidden_size_in]	int8	ND	激活值。
ExpertWeight	[num_experts, hidden_size_in, hidden_size_out]	int8	ND/NZ	专家权重。如果设置“transposeB”为true，则hidden_size_in和hidden_size_out位置调换。
ExpertCount	[num_experts]	int32	ND	专家 Token 数量。
Expertindex	[num_tokens*topK]	int32	ND	专家 Token 索引。
nscale	[num_experts, hidden_size_out]	float	ND	ExpertWeight方向反量化系数。
mscale	[num_tokens]	float	ND	AcTensor方向反量化系数。

- **输出**

参数	维度	数据类型	格式	描述
Result	[num_tokens*Topk, hidden_size_out]	float16/bf16	ND	输出。

- **DOWN**

 版权所有 © 华为技术有限公司

非量化场景

- **输入**

参数	维度	数据类型	格式	描述
AcTensor	[num_tokens*topK, hidden_size_in]	float16/bf16	ND	激活值。
ExpertWeight	[num_experts, hidden_size_in, hidden_size_out]	float16/bf16	ND	专家权重。如果设置“transposeB”为true，则hidden_size_in和hidden_size_out位置调换。
ExpertCount	[num_experts]	int32	ND	专家 Token 数量。
Expertindex	[num_tokens*topK]	int32	ND	专家 Token 索引。

- **输出**

参数	维度	数据类型	格式	描述
Result	[num_tokens, hidden_size_out]	float16/bf16	ND	输出。

量化场景

- **输入**

参数	维度	数据类型	格式	描述
AcTensor	[num_tokens*topK, hidden_size_in]	int8	ND	激活值。
ExpertWeight	[num_experts, hidden_size_in, hidden_size_out]	int8	ND/NZ	专家权重。如果设置“transposeB”为true，则hidden_size_in和hidden_size_out位置调换。
ExpertCount	[num_experts]	int32	ND	专家 Token 数量。
Expertindex	[num_tokens*topK]	int32	ND	专家 Token 索引。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
nscale	[num_experts, hidden_size_out]	float	ND	ExpertWeight方向反量化系数。
mscale	[num_tokens*topK]	float	ND	AcTensor方向反量化系数。

- **输出**

参数	维度	数据类型	格式	描述
Result	[num_tokens, hidden_size_out]	float16/bf16	ND	输出。

规格约束

- 当选择输出类型量化 ，则输入必须为int8类型；当选择输出类型不量化 ，则输入必须为bf16或float16。
- topK的数量不能超过专家数量。
- 只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- nTokens取值范围[128, 512]。
- nExperts取值范围[128, 256]。
- topK取值范围[2, 10]。
- hiddenSizeIn在UP模式下的取值范围[32, 5120]且为32的整数倍，在DOWN模式下取值范围[32, 256]且为32的整数倍。
- hiddenSizeOut在UP模式下的取值范围[32, 256]且为32的整数倍，在DOWN模式下取值范围[32, 5120]且为32的整数倍。

10.30 GroupTopkOperation

功能

GroupTopk算子超参数。将输入tensor0中维度1（输入tensor0有2个维度：维度0和维度1）数据分groupNum个组，每组取最大值，然后选出每组最大值中前k个，最后将非前k个组的数据全部置零。

定义

```
struct GroupTopkParam {
    int32_t groupNum = 1;
    int32_t k = 0;
    enum GroupMultiFlag : uint16_t {
        UNDEFINED = 0,
        SUM_MULTI_MAX
    };
    uint16_t n = 1;
    uint8_t rsv[12] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
groupNum	int32_t	1	[1, expert_num]	否	每个token分组数量。注：“expert_num”为输入参数token维度dim_1的值。
k	int32_t	0	[1, groupNum]	是	选择top K专家数量。需要大于等于1。
groupMultiFlag	uint16_t	UNDEFINED	[0,1]	否	枚举值，组内取值计算类型。 ● UNDEFINED：默认类型，每组内取最大值。 ● SUM_MULTI_MAX：每组内取n个最大值求和，需要设置参数n。
n	uint16_t	1	[1, expert_num/ groupNum]	否	每组内取值的个数。 ● groupMultiFlag为1时，n需要大于0。 ● groupMultiFlag为0时不生效。
rsv[12]	uint8_t	{0}	-	-	预留参数。

输入

参数	维度	数据类型	格式	描述
token	[dim_0, dim_1]	float16/ bf16	ND	输入tensor0， 二维Tensor，dim_0为token数，dim_1为专家总数。
idxArr	[1024]	int32	ND	输入tensor1， 一维Tensor，用于辅助计算，固定长度1024，[0,1,2,...,1023]的等差序列。

输出

参数	维度	数据类型	格式	描述
output	[dim_0, dim_1]	float16/ bf16	ND	输出tensor，只有一个输出tensor，是对输入tensor0原地写的输出。数据类型与输入tensor0保持一致。

 版权所有 © 华为技术有限公司

规格约束

- 1≤expert_num≤1024，expert_num≥groupNum≥k≥1，expert_num能被groupNum整除。
- 只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

基础功能

- **功能概述**

  将输入的各专家分数分组，每组内选取最大值，根据每组最大值大小选取topk组，其余组置零。

- **计算公式**

  对于每个token，首先将token的数据均匀分groupNum组，

  a. 每组取最大值；

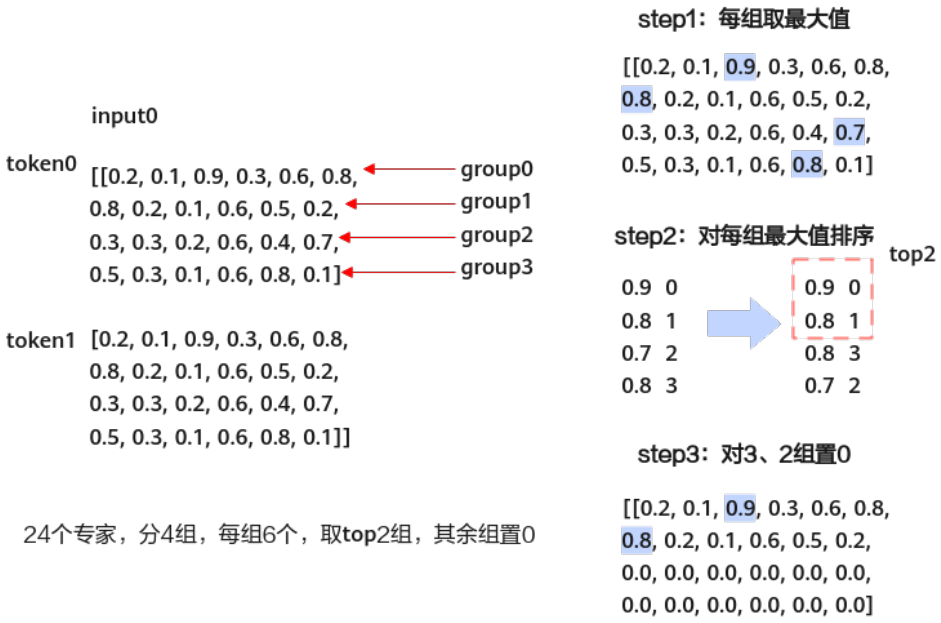
  b. 对每组最大值降序排序；

  c. 获取前k个组的index，然后将其余的index对应的组的数据置零。

- **计算图**

  groupMultiFlag取0时，计算过程下图所示：

  图 10-32 功能示例



- **硬件支持情况**

硬件型号	支持情况
Atlas 推理系列产品	不支持
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持

- **参数列表**

见**参数列表**，需要满足groupMultiFlag置为UNDEFINED。

- **输入**

参数	维度	数据类型	格式	描述
token	[num_tokens, expert_num]	float16/bf16	ND	二维tensor。 • 维度0为token数。 • 维度1为专家总数。
idxArr	[1024]	int32	ND	一维tensor，用于辅助计算，固定长度1024，[0,1,2,...,1023]的等差序列。

- **输出**

参数	维度	数据类型	格式	描述
output	[num_tokens, expert_num]	float16/bf16	ND	是对输入token原地写的输出。数据类型与输入token保持一致。

- **使用示例**

```
// 参数构造
atb::infer::GroupTopkParam param;
param.groupNum = 8;
param.k = 1;
```

组内再排序和选取前 n 个值功能

- **功能概述**

  将输入的各专家分数分组，每组内降序排列选取前n个值求和，根据和的大小选取topk组，其余组置零。

- **计算公式**

  对于每个token，首先将token的数据均匀分groupNum组，

  a. 每组降序排列，取前n个值；
  b. 对这n个值求和；
  c. 各个组根据求得的和大小降序排序；
  d. 获取前k个组的index，然后将其余的index对应的组的数据置零。

  说明

  n取1时对应**基础功能**场景。

- **硬件支持情况**

硬件型号	支持情况
Atlas 推理系列产品	不支持

 版权所有 © 华为技术有限公司

硬件型号	支持情况
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持

- **参数列表**
  见**参数列表**，需要满足groupMultiFlag置为SUM_MULTI_MAX。
- **输入**

参数	维度	数据类型	格式	描述
token	[num_tokens, expert_num]	float16/bf16	ND	二维tensor。 • 维度0为token数。 • 维度1为专家总数。
idxArr	[1024]	int32	ND	一维tensor，用于辅助计算，固定长度1024，[0,1,2,...,1023]的等差序列。

- **输出**

参数	维度	数据类型	格式	描述
output	[num_tokens, expert_num]	float16/bf16	ND	是对输入tensor0原地写的输出。数据类型与输入tensor0保持一致。

- **使用示例**

```
// 参数构造
atb::infer::GroupTopkParam param;
param.groupNum = 8;
param.k = 1;
param.groupMultiFlag = 1;
param.n = 2;
```

10.31 IndexAddOperation

功能

固定维度的指定下标加上某个特定值。

 版权所有 © 华为技术有限公司

图 10-33 IndexAddOperation



定义

```
struct IndexAddParam {
    enum IndexType {
        INDEX_UNDEFINED = 0,
        INDEX_ADD,
        INDEX_ADD_VALID,
    };
    IndexType indexType = INDEX_UNDEFINED;
    int64_t axis = 0;
    uint8_t rsv[16] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
indexType	IndexType	INDEX_UNDEFINED	0，1，2	是	操作类型。 • INDEX_UNDEFINED：未定义。 • INDEX_ADD：加。 • INDEX_ADD_VALID：有效长度内加。不支持Atlas 推理系列产品。
axis	int64_t	0	• indexType = 1时：[-intensor[0].dimNum, intensor[0].dimNum - 1] • indexType = 2时：0	否	输入tensor需加上updates更新值的轴。 • 当“indexType”为INDEX_ADD时，可为负数，取值范围为[-varDimNum, varDimNum - 1]。varDimNum为输入tensor0的维度数。 • 当“indexType”为INDEX_ADD_VALID时，仅支持取值为0。
rsv[16]	uint8_t	{0}	[0]	否	预留参数。

 版权所有 © 华为技术有限公司

输入输出描述（INDEX_ADD）

参数	维度	数据类型	格式	描述
var	[dim_0, ..., dim_n]	float16/ bf16	ND	输入tensor。被加数，输入为零，原地被加后作为输出。
indices	[dim_x]	int32	ND	输入tensor。指定固定维度的指定下标。dim_min = min(dim_x, dim_axis)，值域为[0, dim_min)，且前dim_min个值不重复。
updates	[dim_0, ..., dim_n]	float16/ bf16	ND	输入tensor。加数，根据indices的值加到var对应位置。维度数与var一致。索引为axis的维度为dim_x，即dim_axis == dim_x。
alpha	[1]	float16/ bf16	ND	输入tensor。累加次数。
output	[dim_0, ..., dim_n]	float16/ bf16	ND	输出tensor。与var为同一个Tensor，即二者数据类型、数据格式和地址等所有属性均相同。
min(x, y)表示取x和y两个数中的较小值。 dim_axis表示var、updates和output在索引为axis的维度大小，即：若axis为0，则dim_axis对应第0维大小d_0。				

输入输出描述（INDEX_ADD_VALID）

参数	维度	数据类型	格式	描述
var	[dim_1, dim_2]	float16/ bf16	ND	输入tensor。被加数，输入为零，原地被加后作为输出。
indices	[dim_0]	int32	ND	输入tensor。指定固定维度的指定下标。值域范围为[0, dim_1)。
updates	[dim_0, dim_2]	float16/ bf16	ND	输入tensor。加数，根据indices的值加到var对应位置。
validIndicesNum	[1]	int32	ND	输入tensor。indices的有效长度。值域范围为[0, dim_0]。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
output	[dim_1, dim_2]	float16/ bf16	ND	输出tensor。与var为同一个Tensor，即二者数据类型、数据格式和地址等所有属性均相同。
dim_2取值范围为(0, 8192]。				

接口调用示例

- **indexType = INDEX_ADD**
  - 输入

    indexType = 0

    axis = 0

    var =

    ```
    [[ 1, 1, 1 ],
     [ 1，1，1],
     [ 1，1，1],
     [ 1，1，1],
     [ 1，1，1]]
    ```

    indices = torch.tensor[0, 4, 2]

    updates =

    ```
    [[ 1, 2, 3 ],
     [ 4, 5, 6],
     [ 7, 8, 9]]
    ```

    alpha = 1
  - 输出

    output =

    ```
    [[  2,   3,   4],
     [  1,   1,   1],
     [  8,   9,  10],
     [  1,   1,   1],
     [  5,   6,   7]]
    ```
- **indexType = INDEX_ADD_VALID**
  - 输入

    indexType = 1

    axis = 0

    var =

    ```
    [[ 0, 0, 0 ],
     [ 0，0，0],
     [ 0，0，0],
     [ 0，0，0],
     [ 0，0，0]]
    ```

    indices = {[0, 2, 2, -1, -1]}

    updates =

    ```
    [[ 0, 1, 2 ],
     [ 1, 2, 3],
     [ 2, 3, 4],
     [5, 6, 7],
     [6, 7, 8]]
    ```

 版权所有 © 华为技术有限公司

validIndicesNum = {[3]}

- 输出

output =

```
[[ 0,  1,  2],
 [ 0,  0,  0],
 [ 3,  5,  7],
 [ 0,  0,  0],
 [ 0,  0,  0]]
```

10.32 KVCacheOperation

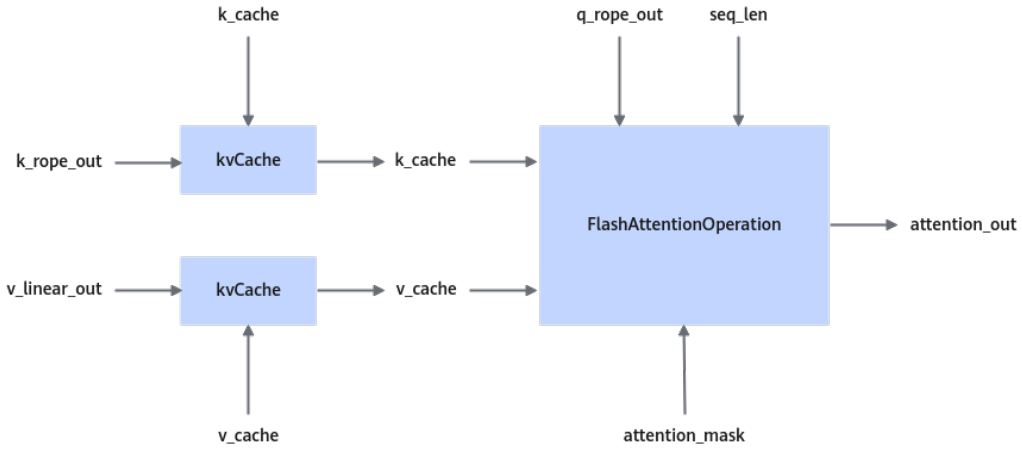
功能

KVCache处理。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

算子上下文

在selfattention算子内部使用，用于将k，v储存到kcache，vcache中。

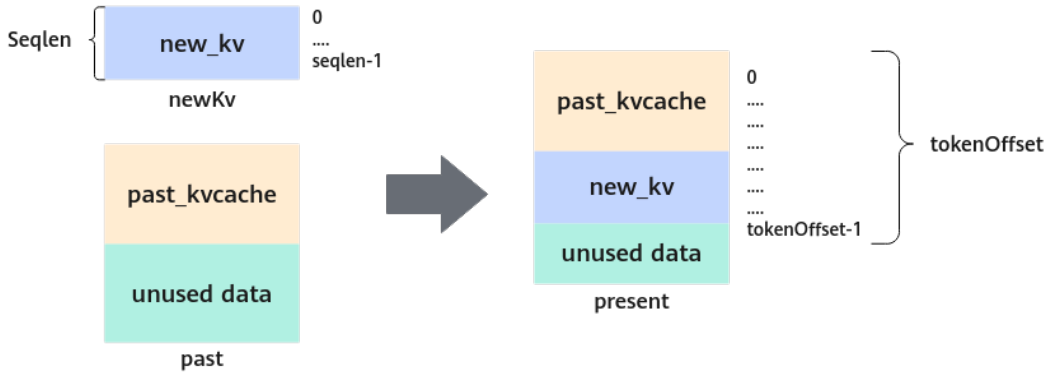
图 10-34 kvCache 在 SelfAttention 中的位置



算子功能实现描述

对每一个batch：

图 10-35 kvCache 算子功能示意图



 版权所有 © 华为技术有限公司

计算逻辑：

```
prefix_ntokens = 0
for i in range(batch):
    for j in range(seqlen[i]):
        cache_out[layer_id[0]][i][token_offset[i] - seqlen[i] + j][:] = newkv[prefix_ntokens + j][:] # 只会修改
layer_id表示的layer的cache内容
    prefix_ntokens += seqlen[i]
```

使用场景

用于transformer推理阶段。

举个例子，用户输入“世界最高峰”，模型续写得到的输出为“是珠穆朗玛峰”，KV Cache每一步的计算过程如下。

- **第一步**生成时，缓存K，V均为空，输入为“世界最高峰”，视为三个单词，模型将按照常规方式并行计算：
  a. 并行计算得到每个token对应的k，v，以及注意力表示 $b^1...b^3$ 。
  b. 使用 b^3 预测下一个token，得到“是”。
  c. 更新缓存，令 $K=[k^1,k^2,k^3]$ ， $V=[v^1,v^2,v^3]$ 。
- **第二步**生成时，计算流程如下：
  a. 仅将“是”输入模型，对其词向量进行映射，得到 q^4 ， k^4 ， v^4 。
  b. 更新缓存，令 $K=[k^1,k^2,k^3,k^4]$ ， $V=[v^1,v^2,v^3,v^4]$ 。
  c. 计算 $b^4=\sum_{i=1}^{4} softmax(q^4*k^i)v^i$ ，预测下一个token，得到“珠”。

定义

```
struct KVCacheParam {
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
newKv	[ntokens, hiddenSize]	float16/ int8	ND	待被cache的key或value。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
layerId	[1]	int32	ND	指定要cache的layer。 算子只会修改past中layer_id表示的layer的cache内容。
past	[layer, batch, maxSeqLen, hiddenSize]	float16/ int8 与newKv保持一致	ND	已经被cache的历史key或value。
tokenOffset	[batch]	int32	ND	每batch上做完cache后的token偏移。
seqLen	[batch]	int32	ND	每batch上newKv的seqlen。

输出

参数	维度	数据类型	格式	描述
present	[layer, batch, maxSeqLen, hiddenSize]	float16/ int8	ND	cache后的key或value，作为输出。 输出present与输入past指向同一地址，即进行原地修改。 数据类型和格式应该与newkv保持一致。

规格约束

- seqLen 数组元素大于0，数组和为nTokens。
- tokenOffset数组元素大于0，小于maxSeqLen。
- layerId要小于past的第一维。
- Atlas 800I A2 推理产品/Atlas A2 训练系列产品上newKv可传四维[batch, seq_len, head_num, head_size]，此时head_num * head_size应该等于hiddenSize。

10.33 LayerNormOperation

功能

LayerNorm是一种归一化方法，可以将网络层输入数据归一化到[0, 1]之间。当前支持三种：NORM、PRENORM、POSTNORM三种归一化。

 版权所有 © 华为技术有限公司

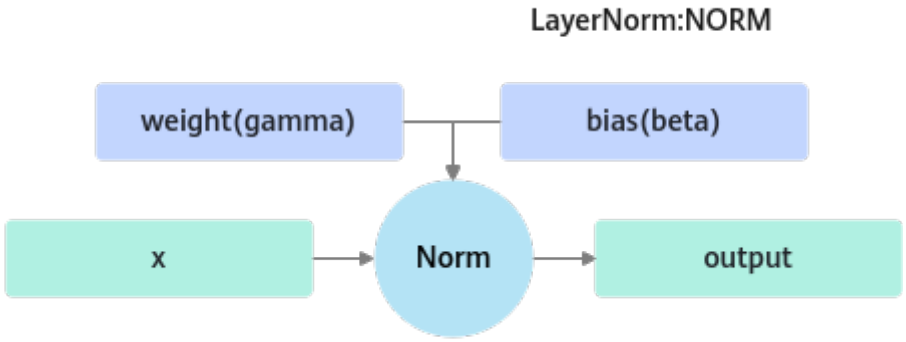
计算公式

- **Norm**

  标准的LayerNorm操作，公式为：

$$y = \frac{x - E[x]}{\sqrt{x + epsilon}} * gamma + beta$$

  图 10-36 LayerNorm 计算公式图



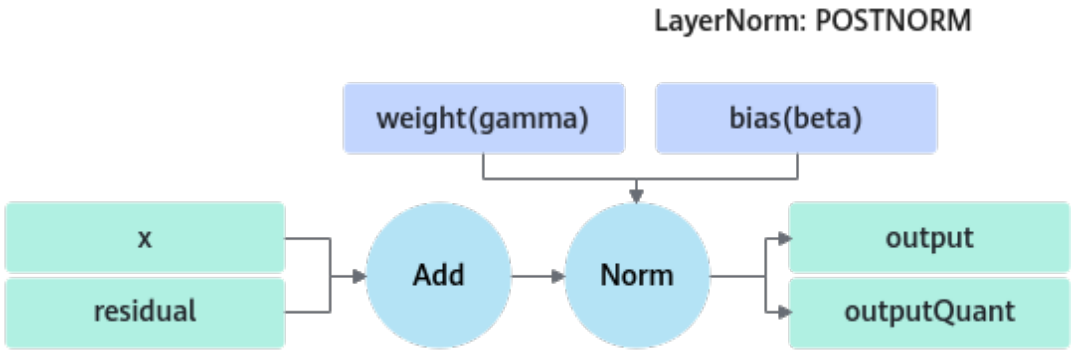
- **PostNorm**

  add和layernorm融合算子，先将x和residual相加再做layernorm归一化，公式为：

$$x' = x + zoomScaleValue * residual$$

$$y = \frac{x' - E[x']}{\sqrt{x' + epsilon}} * gamma + beta$$

  图 10-37 Add+LayerNorm 计算公式图



- **PreNorm**

  add和layernorm融合算子，先将x和residual相加再做layernorm归一化。公式与PostNorm相同。和postNorm区别在于会输出x和residual相加的结果。

 版权所有 © 华为技术有限公司

定义

```
struct LayerNormParam {
    enum LayerNormType : int {
        LAYER_NORM_UNDEFINED = 0,
        LAYER_NORM_NORM,
        LAYER_NORM_PRENORM,
        LAYER_NORM_POSTNORM,
        LAYER_NORM_MAX,
    };
    struct NormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        int32_t beginNormAxis = 0;
        int32_t beginParamsAxis = 0;
        DynamicQuantType dynamicQuantType = DYNAMIC_QUANT_UNDEFINED;
        uint8_t rsv[20] = {0};
    };
    struct PreNormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        uint64_t opMode = 0;
        float zoomScaleValue = 1.0f;
        uint8_t rsv[20] = {0};
    };
    struct PostNormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        uint64_t opMode = 0;
        float zoomScaleValue = 1.0f;
        uint8_t rsv[20] = {0};
    };
    LayerNormType layerType = LAYER_NORM_UNDEFINED;
    NormParam normParam;
    PreNormParam preNormParam;
    PostNormParam postNormParam;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	描述
layerType	归一化类型。 • LAYER_NORM_UNDEFINED：默认值，未定义。 • LAYER_NORM_NORM：norm类型。 • LAYER_NORM_PRENORM：prenorm类型。 • LAYER_NORM_POSTNORM：postnorm类型。
normParam	NORM参数，具体请参见**表10-3**。
preNormParam	PRETNORM参数，具体请参见**表10-4**。
postNormParam	POSTNORM参数，具体请参见**表10-5**。
rsv[8]	预留参数。

 版权所有 © 华为技术有限公司

表 10-3 normParam 成员

参数	类型	默认值	描述
quantType	QuantType	QUANT_UNQUANT	量化类型。 当前支持以下类型。 ● QUANT_UNQUANT。 ● QUANT_INT8。
epsilon	float	1e-5	Epsilon，归一化时加在分母上防止除零。
beginNormAxis	int32_t	0	归一化的维度，默认值为0，从第几维开始norm，同时决定输入gamma和beta维度。
beginParamsAxis	int32_t	0	归一化的维度，默认值为0，决定从第几维开始把后面的维度按轴合并。
dynamicQuantType	DynamicQuantType	DYNAMIC_QUANT_UNDEFINED	动态量化类型。默认为DYNAMIC_QUANT_UNDEFINED非动态量化。当前版本暂不支持非对称动态量化。
rsv[20]	uint8_t	{0}	预留参数。

说明

8.0.RC2及后续版本normParam不再支持quantInputScale、quantInputOffset、quantInputAlpha量化参数。

表 10-4 preNormParam 成员

参数	类型	默认值	描述
quantType	QuantType	QUANT_UNQUANT	量化类型。当前仅支持QUANT_UNQUANT。
epsilon	float	1e-5	Epsilon，归一化时加在分母上防止除零。
opMode	uint64_t	0	● 0：高精度。 ● 1：高性能。（ 暂不支持 ）
zoomScaleValue	float	1.0f	缩放因子。
rsv[20]	uint8_t	{0}	预留参数。

 版权所有 © 华为技术有限公司

表 10-5 postNormParam 成员

参数	类型	默认值	描述
quantType	QuantType	QUANT_UNQUANT	量化类型。当前支持以下类型。 ● QUANT_UNQUANT。 ● QUANT_INT8。
epsilon	float	1e-5	Epsilon，归一化时加在分母上防止除零。
opMode	uint64_t	0	● 0：高精度。 ● 1：高性能。（暂不支持）
zoomScaleValue	float	1.0f	缩放因子。
rsv[20]	uint8_t	{0}	预留参数。

说明

8.0.RC2及后续版本postNormParam不再支持quantInputScale、quantInputOffset、quantInputAlpha量化参数。

LAYER_NORM_NORM 输入输出

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND	输入。 ● quantType = QUANT_INT8时，数据类型为float16或bf16，最后一维的大小要32字节对齐。 ● quantType = QUANT_UNQUANT时，数据类型为float16、float或bf16。
gamma	[beginNormAxis:]或[1, ..., dim_n]	float16/float/bf16	ND	输入。 ● quantType = QUANT_UNQUANT时，根据beginNormAxis确定其余维度。 ● quantType = QUANT_INT8时，维度为[dim_n]，最后一维的大小要32字节对齐。dim_n前可以有任意个1。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
beta	[beginNormAxis:]或[1, ..., dim_n]	float16/float/bf16	ND	输入。 • quantType = QUANT_UNQUANT时，根据beginNormAxis确定其余维度。 • quantType = QUANT_INT8时，维度为[dim_n]，最后一维的大小要32字节对齐。dim_n前可以有任意个1。维度需与gamma保持一致。
scale	[1]	float16/bf16	ND	输入。quantType = QUANT_INT8时才输入。
offset	[1]	int8	ND	输入。quantType = QUANT_INT8时才输入。
output	[dim_0, dim_1, ..., dim_n]	float16/float/int8/bf16	ND	输出。 • quantType = QUANT_INT8时，数据类型为int8。 • quantType = QUANT_UNQUANT时，数据类型为float16、float或bf16。

LAYER_NORM_NORM DYNAMIC_QUANT 输入输出

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16	ND	输入。最后一维n的大小要32字节对齐，且小于等于12288。
gamma	[dim_n]	float16	ND	输入。最后一维n的大小要32字节对齐，且小于等于12288。
beta	[dim_n]	float16	ND	输入。所有属性与gamma一致。最后一维n的大小要32字节对齐，且小于等于12288。
y	[dim_0, dim_1, ..., dim_n]	int8	ND	输出。最后一维n的大小要32字节对齐，且小于等于12288。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
scale	[dim_0，dim_1，... ，dim_n-1]	float	ND	输出。当quantType == QUANT_INT8且dynamicQuantType != DYNAMIC_QUANT_UNDEFINED时输出。
offset	[dim_0，dim_1，... ，dim_n-1]	float	ND	输出。当quantType == QUANT_INT8且dynamicQuantType == DYNAMIC_QUANT_ASYMMETRIC时输出。当前版本暂不支持。

LAYER_NORM_PRENORM 输入输出

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输入。最后一维的大小要32字节对齐。
residual_in	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输入。最后一维的大小要32字节对齐。
gamma	[1, ..., dim_n]	float16/bf16	ND	输入。最后一维的大小要32字节对齐。dim_n前可以有任意个1。
beta	[1, ..., dim_n]	float16/bf16	ND	输入。最后一维的大小要32字节对齐。dim_n前可以有任意个1。维度需要与gamma保持一致。
output	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输出。
residual_out	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输出。Norm前的输出（等于x + residual_in * zoom_scale）。

LAYER_NORM_POSTNORM 输入输出

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16	ND	输入。最后一维的大小要32字节对齐。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
residual	[dim_0, dim_1, ..., dim_n]	float16	ND	输入。最后一维的大小要32字节对齐。
gamma	[1, ..., dim_n]	float16	ND	输入。最后一维的大小要32字节对齐。dim_n前可以有任意个1。
beta	[1, ..., dim_n]	float16	ND	输入。最后一维的大小要32字节对齐。dim_n前可以有任意个1。维度需要与gamma保持一致。
scale	[1]	float16	ND	输入。当quantType = QUANT_INT8时输入。
offset	[1]	int8	ND	输入。当quantType = QUANT_INT8时输入。
output	[dim_0, dim_1, ..., dim_n]	float16	ND	输出。
outputQuant	[dim_0, dim_1, ..., dim_n]	int8	ND	输出。当quantType = QUANT_INT8时输出。

规格约束

- beginNormAxis 需要小于 tensor x的维度大小。为负数时，与tensor x维度大小之和需要大于0。
- Norm的动态量化只支持DYNAMIC_QUANT_SYMMETRIC对称动态量化。最后一维需要小于等于12288。
- 除量化场景下的scale和offset tensor，其余tensor的最后一维大小相等。

功能列表

- 非量化
  - 基础layernorm
  - add+layernorm融合（prenorm，postnorm）
- 量化
  - layernorm量化
  - layernorm对称动态量化
  - add+layernorm量化（postnorm）

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景

```
#include <iostream>
#include <vector>
```

 版权所有 © 华为技术有限公司

```
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const int32_t DEVICE_ID = 0;
const uint32_t DIM_0 = 128;
const uint32_t DIM_1 = 256;
const int32_t BEGIN_NORM_AXIS = 1;

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    atb::Tensor x = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>(DIM_0 * DIM_1, 2.0),
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {DIM_0, DIM_1});
    atb::Tensor gamma = CreateTensorFromVector(
        contextPtr, stream, std::vector<float>(DIM_1, 2.0), ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{DIM_1});
    atb::Tensor beta = CreateTensorFromVector(
        contextPtr, stream, std::vector<float>(DIM_1, 1.0), ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{DIM_1});
    atb::SVector<atb::Tensor> inTensors = {x, gamma, beta};
    return inTensors;
}

/**
 * @brief 创建一个beginNormAxis为1的layernorm operation
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *CreateLayerNormOperation()
{
    atb::infer::LayerNormParam param;
    param.layerType = atb::infer::LayerNormParam::LayerNormType::LAYER_NORM_NORM;
    param.normParam.beginNormAxis = BEGIN_NORM_AXIS;
    atb::Operation *layerNormOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &layerNormOp));
    return layerNormOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    atb::Context *context = nullptr;
    void *stream = nullptr;

    CHECK_STATUS(aclInit(nullptr));
    CHECK_STATUS(aclrtSetDevice(DEVICE_ID));
    CHECK_STATUS(atb::CreateContext(&context));
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 创建op
    atb::Operation *layerNormOp = CreateLayerNormOperation();
    // 准备输入tensor
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
```

 版权所有 © 华为技术有限公司

```
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {DIM_0, DIM_1});
    variantPack.outTensors = {tensorOut};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspace大小
    CHECK_STATUS(layerNormOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // layernorm执行
    layerNormOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放资源
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    CHECK_STATUS(aclrtFree(tensorOut.deviceData));
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(layerNormOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    std::cout << "Layernorm demo success!" << std::endl;
    return 0;
}
```

10.34 LayerNormWithStrideOperation

功能

为了使LayerNorm支持非连续Tensor，将x tensor通过（shape、strides、offset）表示。strides描述某个维度上相邻两个元素的间隔，offset表示这个Tensor的首元素相对addr的偏移。

硬件支持情况

硬件型号	支持情况
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持

定义

```
struct LayerNormWithStrideParam {
    enum LayerNormType : int {
        LAYER_NORM_UNDEFINED = 0,
        LAYER_NORM_NORM
    };
    struct NormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        int32_t beginNormAxis = 0;
        int32_t beginParamsAxis = 0;
        DynamicQuantType dynamicQuantType = DYNAMIC_QUANT_UNDEFINED;
        uint8_t rsv[20] = {0};
    };
```

 版权所有 © 华为技术有限公司

```
    LayerNormType layerType = LAYER_NORM_UNDEFINED;
    NormParam normParam;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	描述
layerType	归一化类型。 • LAYER_NORM_UNDEFINED：默认值，未定义。 • LAYER_NORM_NORM：norm类型。
normParam	norm参数，具体请参见**表1 normParam成员**。
rsv[8]	预留参数。

表 10-6 normParam 成员

成员名称	类型	默认值	取值范围	是否必选	描述
quantType	QuantType	QUANT_UNDEFINED	QUANT_UNDEFINED	否	量化类型QuantType。当前仅支持QUANT_UNDEFINED。
epsilon	float	1e-5	大于0	否	Epsilon，归一化时加在分母上防止除0。
beginNormAxis	int32_t	0	beginNormAxis 需要小于 tensor x 的维度大小。为负数时，与 tensor x 维度大小之和需要大于0。	否	归一化的维度，从第几维开始norm，同时决定gamma和beta的维度。不启用，只支持0。
beginParamAxis	int32_t	0	beginParamAxis 需要大于等于0小于 tensor x的维度大小。	否	归一化的维度，决定从第几维开始把后面的维度按轴合并。不启用，只支持0。
dynamicQuantType	DynamicQuantType	DYNAMIC_QUANT_UNDEFINED	DYNAMIC_QUANT_UNDEFINED	否	只支持DYNAMIC_QUANT_UNDEFINED。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/ bf16	ND	最后一维的大小要32字节对齐。
gamma	[1, dim_n]	float16/ bf16	ND	最后一维的大小要32字节对齐。
beta	[1, dim_n]	float16/ bf16	ND	最后一维的大小要32字节对齐。
xStrides	[n+1]	int32	ND	x的stride步长，最后一维的值为1，x_shape表示x的维度数，传入方式为host tensor，大于0，最后一个数需为1。
xOffset	[1]	int32	ND	x的offset，需大于0，默认值为0，传入方式为host tensor。

输出

参数	维度	数据类型	格式	描述
output	[dim_0, dim_1, ..., dim_n]	float16/ bf16	ND	输出tensor，与x的维度保持一致。

10.35 LinearOperation

10.35.1 功能

将A、B两个矩阵进行矩阵乘运算，同时可以选择对矩阵乘的运算结果添加偏置、InplaceAdd融合或反量化操作。

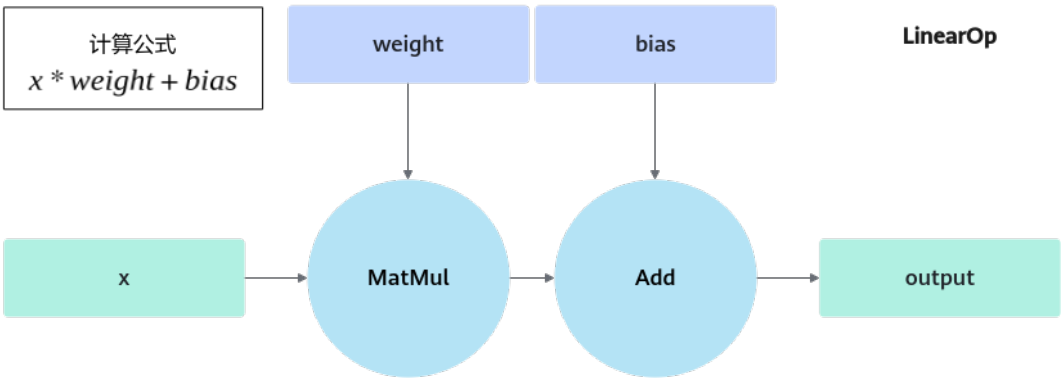
算子本质上是接收x和weight两个输入tensor作为A矩阵和B矩阵进行矩阵乘运算，可通过参数“transposeA”与“transposeB”控制做矩阵乘前是否需要对A矩阵和B矩阵进行行列转置，根据参数转置后的A矩阵和B矩阵需满足矩阵乘维度关系，例如，当“transposeA”为false，“transposeB”为true时，x和weight的shape可以分别为[m, k]和[n, k]。

该算子支持浮点和量化场景，当参数“outDataType”值为ACL_DT_UNDEFINED时为浮点场景，否则为量化场景。

该算子支持爱因斯坦乘场景，当参数“matmulType”值为MATMUL_EIN_SUM为爱因斯坦乘场景。

 版权所有 © 华为技术有限公司

图 10-38 LinearOperation



10.35.2 对标算子

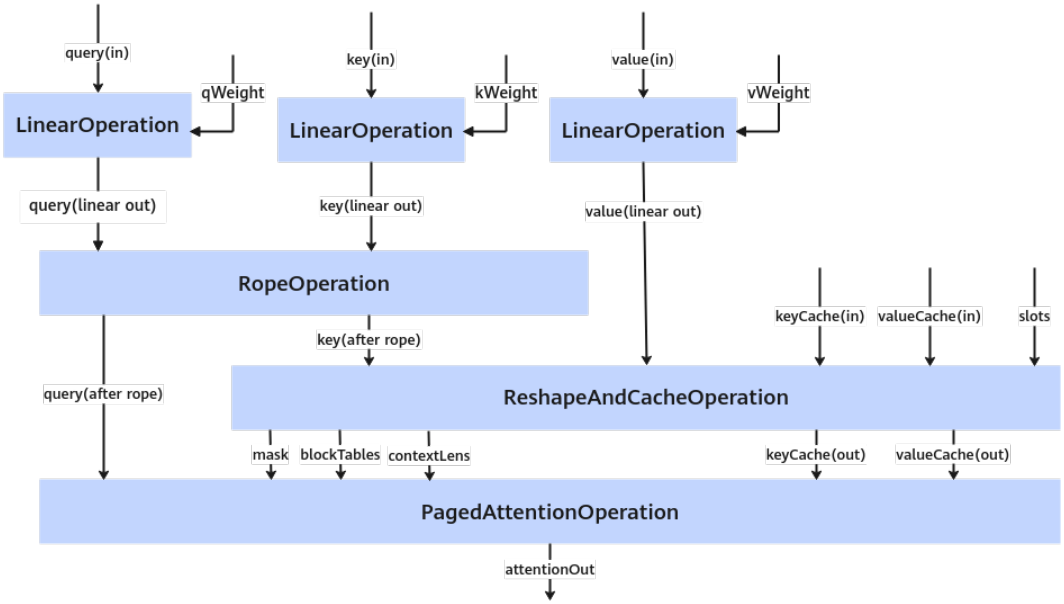
本OP对标torch库中的matmul算子，功能略有不同。部分对标的功能参考**torch.matmul文档**中的示例：

```
# batched matrix x batched matrix
tensor1 = torch.randn(10, 3, 4)
tensor2 = torch.randn(10, 4, 5)
torch.matmul(tensor1, tensor2).size() # torch.Size([10, 3, 5])
# batched matrix x broadcasted matrix
tensor1 = torch.randn(10, 3, 4)
tensor2 = torch.randn(4, 5)
torch.matmul(tensor1, tensor2).size() # torch.Size([10, 3, 5])
```

10.35.3 算子上下文

在transformer模型中，矩阵乘被大量使用，其中，将query、key、value分别和权重进行矩阵乘是最典型的使用场景，下图为linear在模型中的位置示意。

图 10-39 LinearOperation 算子上下文



 版权所有 © 华为技术有限公司

10.35.4 定义

```
struct LinearParam {
    enum MatmulType : uint8_t {
        MATMUL_UNDEFINED = 0,
        MATMUL_EIN_SUM,
    }
    bool transposeA = false;
    bool transposeB = true;
    bool hasBias = true;
    aclDataType outDataType = ACL_DT_UNDEFINED;
    bool enAccum = false;
    MatmulType matmulType = MATMUL_UNDEFINED;
    uint8_t rsv[22] = {0};
};
```

10.35.5 参数列表

成员名称	类型	默认值	取值范围	描述
transposeA	bool	false	false/true	是否需要对A矩阵进行行列转置。取值为true时进行转置，否则不转置。
transposeB	bool	true	false/true	是否需要对B矩阵进行行列转置。取值为true时进行转置，否则不转置。
hasBias	bool	true	false/true	是否叠加偏置。取值为true时叠加偏置，否则不叠加。
outDataType	aclDataType	ACL_DT_UNDEFINED	ACL_DT_UNDEFINED/ ACL_FLOAT16/ ACL_BF16	指定输出tensor的数据类型。默认值表示根据输入tensors自动推导输出tensors数据类型。
enAccum	bool	false	false/true	是否使能累加，即matmul+inplace add融合场景。
matmulType	MatmulType	MATMUL_UNDEFINED	MATMUL_UNDEFINED/ MATMUL_EIN_SUM	矩阵乘功能类型。 • MATMUL_UNDEFINED：包含除了爱因斯坦乘的其他特性。 • MATMUL_EIN_SUM：使能爱因斯坦乘。
rsv[22]	uint8_t	{0}	[0]	预留参数。

详细说明

transposeA和transposeB

- 根据transposeA和transposeB的配置，转置后的A矩阵和B矩阵shape需满足矩阵乘维度关系；
- transposeA和transposeB的取值在部分功能中存在约束；

 版权所有 © 华为技术有限公司

- transposeA和transposeB可用于提升特殊场景下算子性能，收益根据shape动态变化。以常规矩阵乘为例，假设A矩阵和B矩阵的shape分别为[m, k]和[k, n]：
  - 当m为256的整数倍且k不为256的整数倍时，配置transposeA为true，将A矩阵进行行列转置，可提升性能，此时A矩阵shape为[k, m]；
  - 当k为256的整数倍且n不为256的整数倍时，配置transposeB为true，将B矩阵进行行列转置，可提升性能，此时B矩阵shape为[n, k]。
- 后文进行功能描述时，默认为transposeA与transposeB取值均为false的情况，维度对应关系如下表：

A矩阵原维度	transposeA为true时A矩阵维度
[m, k]	[k, m]
[batch, m, k]	[batch, k, m]

B矩阵原维度	transposeB为true时B矩阵维度
[k, n]	[n, k]
[batch, k, n]	[batch, n, k]
[1, n / 16, k, 16]	[1, k / 16, n, 16]
[batch, n / 16, k, 16]	[batch, k / 16, n, 16]
[1, n / 32, k, 32]	[1, k / 32, n, 32]
[batch, n / 32, k, 32]	[batch, k / 32, n, 32]

hasBias

- 当enAccum配置为true时，hasBias仅支持配置为false。
- hasBias的取值在部分功能中存在约束。

outDataType

在反量化场景中，配置为输出tensor的数据类型，目前支持ACL_FLOAT16/ACL_BF16；否则，在其他场景中，配置为默认值ACL_DT_UNDEFINED。

enAccum

- 当hasBias配置为true时，enAccum仅支持配置为false。
- 该参数配置为true时，触发matmul + inplaceAdd融合功能，该功能具体设计与约束参考**10.35.6.3 matmul与inplaceAdd融合**。

matmulType

- 在爱因斯坦乘场景中，配置为MATMUL_EIN_SUM，在其他场景中配置为MATMUL_UNDEFINED。
- 当matmulType配置为MATMUL_EIN_SUM时
  - transposeA仅支持配置为false。

 版权所有 © 华为技术有限公司

- hasBias仅支持配置为false。
- enAccum仅支持配置为false。
- outDataType仅支持配置为默认值ACL_DT_UNDEFINED。

10.35.6 功能列表

10.35.6.1 常规 matmul 与叠加偏置

功能概述

常规矩阵乘matmul的功能，且可通过hasBias参数控制是否叠加偏置。

计算公式

矩阵乘输入两个张量A（x）和B（weight），输出张量为C：

$$C = A * B$$

叠加偏置，偏置矩阵为bias：

$$C = A * B + bias$$

硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 200I/500 A2 推理产品	支持	不支持输入输出tensor数据类型为bf16的场景。
Atlas 推理系列产品	支持	-
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

参数配置

成员名称	取值范围	特殊说明
transposeA	false/true	取值为true时，不支持部分场景，详见**规格说明**。
transposeB	false/true	-
hasBias	false/true	-
outDataType	ACL_DT_UNDEFINED	-

 版权所有 © 华为技术有限公司

成员名称	取值范围	特殊说明
enAccum	false	-
matmulType	MATMUL_UNDEFINED	-

输入

参数	维度	数据类型	格式	描述
x	[m, k]/[batch, m, k]	float16/bf16	ND	矩阵乘的A矩阵。
weight	[k, n]/[batch, k, n]/[1, n / 16, k, 16]/[batch, n / 16, k, 16]	float16/bf16	ND/NZ	矩阵乘的B矩阵，权重。
bias	[1, n]/[n]/[batch, n]	float16/bf16	ND	叠加的偏置矩阵。当hasBias取值为true时输入。

输出

参数	维度	数据类型	格式	描述
output	[m, n]/[batch, m, n]	float16/bf16	ND	矩阵乘计算结果。

规格说明

由于输入输出的排列组合约束较复杂，下图列举了所有输入输出属性的组合，图中没有的组合即不支持：

 版权所有 © 华为技术有限公司

图 10-40 输入输出属性排列组合

	x			weight			bias			output		
组合ID	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度
1	float16	ND	[m, k]	float16	ND	[k, n]	float16	ND	[1, n]/[n]	float16	ND	[m, n]
2	float16	ND	[batch, m, k]	float16	ND	[k, n]	float16	ND	[1, n]/[n]	float16	ND	[batch, m, n]
3	float16	ND	[batch, m, k]	float16	ND	[batch, k, n]	float16	ND	[batch, n]	float16	ND	[batch, m, n]
4	float16	ND	[m, k]	float16	NZ	[k, n]/[1, n / 16, k, 16]	float16	ND	[1, n]/[n]	float16	ND	[m, n]
5	float16	ND	[batch, m, k]	float16	NZ	[k, n]/[1, n / 16, k, 16]	float16	ND	[1, n]/[n]	float16	ND	[batch, m, n]
6	float16	ND	[batch, m, k]	float16	NZ	[batch, k, n]/[batch, n / 16, k, 16]	float16	ND	[batch, n]	float16	ND	[batch, m, n]
7	bf16	ND	[m, k]	bf16	ND	[k, n]	bf16	ND	[1, n]/[n]	bf16	ND	[m, n]
8	bf16	ND	[batch, m, k]	bf16	ND	[k, n]	bf16	ND	[1, n]/[n]	bf16	ND	[batch, m, n]
9	bf16	ND	[batch, m, k]	bf16	ND	[batch, k, n]	bf16	ND	[batch, n]	bf16	ND	[batch, m, n]
10	bf16	ND	[m, k]	bf16	NZ	[k, n]/[1, n / 16, k, 16]	bf16	ND	[1, n]/[n]	bf16	ND	[m, n]
11	bf16	ND	[batch, m, k]	bf16	NZ	[k, n]/[1, n / 16, k, 16]	bf16	ND	[1, n]/[n]	bf16	ND	[batch, m, n]
12	bf16	ND	[batch, m, k]	bf16	NZ	[batch, k, n]/[batch, n / 16, k, 16]	bf16	ND	[batch, n]	bf16	ND	[batch, m, n]

- Atlas 推理系列产品不支持组合7-12。
- “transposeA”取值为true时，不支持组合2、5、8、11。
- 所有输入输出tensor的数据类型均相同。
- 当weight维度为4维时，k和n的值均为16的整数倍。

OP 使用与典型场景

OP使用时，可参考算子使用指导中的使用流程部分，其中，单算子（OpsOperation）构造Operation参数的构造方法参考下列各场景的参数构造部分。

- **场景一**

```
// 参数构造
atb::infer::LinearParam param;
param.transposeA = false;
param.transposeB = false;
param.hasBias = false;
param.outDataType = ACL_DT_UNDEFINED;
param.enAccum = false;
param.matmulType = MATMUL_UNDEFINED;
# 计算示例
>>> x
tensor([[1, 2],
        [3, 4]])
>>> weight
tensor([[1, 2, 3],
        [4, 5, 6]])
>>> output
tensor([[9, 12, 15],
        [19, 26, 33]])
# 9  = 1 * 1 + 2 * 4
# 12 = 1 * 2 + 2 * 5
```

 版权所有 © 华为技术有限公司

```
# 15 = 1 * 3 + 2 * 6
# 19 = 3 * 1 + 4 * 4
# 26 = 3 * 2 + 4 * 5
# 33 = 3 * 3 + 4 * 6
```

- **场景二**

```
// 参数构造
atb::infer::LinearParam param;
param.transposeA = true;
param.transposeB = true;
param.hasBias = false;
param.outDataType = ACL_DT_UNDEFINED;
param.enAccum = false;
param.matmulType = MATMUL_UNDEFINED;
# 根据行列转置情况，该示例和前一示例计算相同。
>>> x
tensor([[1, 3],
        [2, 4]])
>>> weight
tensor([[1, 4],
        [2, 5],
        [3, 6]])
>>> output
tensor([[9, 12, 15],
        [19, 26, 33]])
```

- **场景三**

```
// 参数构造
atb::infer::LinearParam param;
param.transposeA = false;
param.transposeB = false;
param.hasBias = true;
param.outDataType = ACL_DT_UNDEFINED;
param.enAccum = false;
param.matmulType = MATMUL_UNDEFINED;
# 计算示例
>>> x
tensor([[1, 2],
        [3, 4]])
>>> weight
tensor([[1, 2, 3],
        [4, 5, 6]])
>>> bias
tensor([1, 2, 3])
>>> output
tensor([[10, 14, 18],
        [20, 28, 36]])
# 10 = 1 * 1 + 2 * 4 + 1
# 14 = 1 * 2 + 2 * 5 + 2
# 18 = 1 * 3 + 2 * 6 + 3
# 20 = 3 * 1 + 4 * 4 + 1
# 28 = 3 * 2 + 4 * 5 + 2
# 36 = 3 * 3 + 4 * 6 + 3
```

功能约束

- “transposeA”为true时不支持部分场景。
- “outDataType”为ACL_DT_UNDEFINED。
- “enAccum”为false。

10.35.6.2 matmul 与 add 融合

功能概述

矩阵乘matmul与add融合，该功能类似功能1中的叠加偏置，性能较优。

 版权所有 © 华为技术有限公司

计算公式

矩阵乘输入两个张量A（x）和B（weight），偏置矩阵为bias，输出张量为C：

$$C = A * B + bias$$

硬件支持情况

硬件型号	支持情况	说明
Atlas 200I/500 A2 推理产品	支持	不支持输入输出tensor数据类型为bf16的场景。
Atlas 推理系列产品	不支持	-
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

参数配置

成员名称	取值范围	特殊说明
transposeA	false/true	取值为true时，不支持部分场景，详见**规格说明**。
transposeB	false/true	-
hasBias	true	-
outDataType	ACL_DT_UNDEFINED	-
enAccum	false	-
matmulType	MATMUL_UNDEFINED	-

输入

参数	维度	数据类型	格式	描述
x	[m, k]/[batch, m, k]	float16/bf16	ND	矩阵乘的A矩阵。
weight	[k, n]/[batch, k, n]	float16/bf16	ND	矩阵乘的B矩阵，权重。
bias	[1, n]/[n]/[batch, n]	float	ND	叠加的偏置矩阵。

 版权所有 © 华为技术有限公司

输出

参数	维度	数据类型	格式	描述
output	[m, n]/[batch, m, n]	float16/bf16	ND	矩阵乘计算结果。

规格说明

图 10-41 输入输出属性排列组合

	x			weight			bias			output		
组合ID	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度
1	float16	ND	[m, k]	float16	ND	[k, n]	float	ND	[1, n]/[n]	float16	ND	[m, n]
2	float16	ND	[batch, m, k]	float16	ND	[k, n]	float	ND	[1, n]/[n]	float16	ND	[batch, m, n]
3	float16	ND	[batch, m, k]	float16	ND	[batch, k, n]	float	ND	[batch, n]	float16	ND	[batch, m, n]
4	bf16	ND	[m, k]	bf16	ND	[k, n]	float	ND	[1, n]/[n]	bf16	ND	[m, n]
5	bf16	ND	[batch, m, k]	bf16	ND	[k, n]	float	ND	[1, n]/[n]	bf16	ND	[batch, m, n]
6	bf16	ND	[batch, m, k]	bf16	ND	[batch, k, n]	float	ND	[batch, n]	bf16	ND	[batch, m, n]

“transposeA”取值为true时，不支持组合2、5。

OP 使用与典型场景

OP使用时，可参考算子使用指导中的使用流程部分，其中，单算子（OpsOperation）构造Operation参数的构造方法参考以下参数构造部分。

```
// 参数构造
atb::infer::LinearParam param;
param.transposeA = false;
param.transposeB = false;
param.hasBias = true;
param.outDataType = ACL_DT_UNDEFINED;
param.enAccum = false;
param.matmulType = MATMUL_UNDEFINED;
# 计算示例
>>> x
tensor([[1, 2],
        [3, 4]])
>>> weight
tensor([[1, 2, 3],
        [4, 5, 6]])
>>> bias
tensor([1, 2, 3])
>>> output
tensor([[10, 14, 18],
        [20, 28, 36]])
# 10 = 1 * 1 + 2 * 4 + 1
# 14 = 1 * 2 + 2 * 5 + 2
# 18 = 1 * 3 + 2 * 6 + 3
# 20 = 3 * 1 + 4 * 4 + 1
# 28 = 3 * 2 + 4 * 5 + 2
# 36 = 3 * 3 + 4 * 6 + 3
```

 版权所有 © 华为技术有限公司

功能约束

- “transposeA”为true时不支持部分场景。
- “hasBias”为true。
- “outDataType”为ACL_DT_UNDEFINED。
- “enAccum”为false。

10.35.6.3 matmul 与 inplaceAdd 融合

功能概述

矩阵乘matmul与inplaceAdd融合，将矩阵乘结果累加到累加矩阵上，相较于算子拼接的方案，性能较优。

计算公式

矩阵乘输入两个张量A（x）和B（weight），累加张量为C（accum），同时也是输出张量：

$$C = A * B + C$$

硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 200I/500 A2 推理产品	不支持	-
Atlas 推理系列产品	不支持	-
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

参数配置

成员名称	取值范围	特殊说明
transposeA	false/true	取值为true时，不支持部分场景，详见**规格说明**。 当输入数据类型为float16时，不支持transposeA为true。
transposeB	false/true	-
hasBias	false	-
outDataType	ACL_DT_UNDEFINED	-
enAccum	true	-

 版权所有 © 华为技术有限公司

成员名称	取值范围	特殊说明
matmulType	MATMUL_UNDEFINED	-

输入

参数	维度	数据类型	格式	描述
x	[m, k]/[batch, m, k]	float16/bf16	ND	矩阵乘的A矩阵。
weight	[k, n]/[batch, k, n]	float16/bf16	ND	矩阵乘的B矩阵，权重。
accum	[m, n]/[batch, m, n]	float	ND	累加矩阵。

输出

参数	维度	数据类型	格式	描述
output	[m, n]/[batch, m, n]	float	ND	累加矩阵，与accum为同一个Tensor，即计算结果原地写。

规格说明

由于输入输出的排列组合约束较复杂，下表列举了所有输入输出属性的组合，表格中没有的组合即不支持：

图 10-42 输入输出属性排列组合

	x			weight			accum			output		
组合ID	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度
1	float16	ND	[m, k]	float16	ND	[k, n]	float	ND	[m, n]	float	ND	[m, n]
2	float16	ND	[batch, m, k]	float16	ND	[k, n]	float	ND	[batch, m, n]	float	ND	[batch, m, n]
3	float16	ND	[batch, m, k]	float16	ND	[batch, k, n]	float	ND	[batch, m, n]	float	ND	[batch, m, n]
4	bf16	ND	[m, k]	bf16	ND	[k, n]	float	ND	[m, n]	float	ND	[m, n]
5	bf16	ND	[batch, m, k]	bf16	ND	[k, n]	float	ND	[batch, m, n]	float	ND	[batch, m, n]
6	bf16	ND	[batch, m, k]	bf16	ND	[batch, k, n]	float	ND	[batch, m, n]	float	ND	[batch, m, n]

 版权所有 © 华为技术有限公司

“transposeA”取值为true时，不支持组合2、5、8、11。

OP 使用与典型场景

OP使用时，可参考算子使用指导中的使用流程部分，其中，单算子（OpsOperation）构造Operation参数的构造方法参考以下参数构造部分。

```
// 参数构造
atb::infer::LinearParam param;
param.transposeA = false;
param.transposeB = false;
param.hasBias = false;
param.outDataType = ACL_DT_UNDEFINED;
param.enAccum = true;
param.matmulType = MATMUL_UNDEFINED;
# 计算示例
>>> x
tensor([[1, 2],
        [3, 4]])
>>> weight
tensor([[1, 2, 3],
        [4, 5, 6]])
>>> accum
tensor([[1, 2, 3],
        [4, 5, 6])
>>> output
tensor([[10, 14, 18],
        [23, 31, 39]])
# 10 = 1 * 1 + 2 * 4 + 1
# 14 = 1 * 2 + 2 * 5 + 2
# 18 = 1 * 3 + 2 * 6 + 3
# 23 = 3 * 1 + 4 * 4 + 4
# 31 = 3 * 2 + 4 * 5 + 5
# 39 = 3 * 3 + 4 * 6 + 6
```

功能约束

- “transposeA”为true时不支持部分场景。
- “hasBias”为false。
- “outDataType”为ACL_DT_UNDEFINED。
- “enAccum”为true。

10.35.6.4 matmul 反量化与叠加偏置

功能概述

矩阵乘matmul加反量化处理，通过hasBias参数控制是否叠加偏置。

计算公式

矩阵乘输入两个张量A（x）和B（weight），反量化步长为deqScale，输出张量为C，dequant为反量化处理函数：

$$C = dequant(A * B, deqScale)$$

叠加偏置的场景下，偏置矩阵为bias：

$$C = dequant(A * B + bias, deqScale)$$

 版权所有 © 华为技术有限公司

硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 200I/500 A2 推理产品	支持	不支持输入输出tensor数据类型为bf16的场景。
Atlas 推理系列产品 Atlas 训练系列产品	支持	• 不支持transposeA配置为true。 • 不支持transposeB配置为false。 • 不支持hasBias为false。 • 不支持outDataType为ACL_BF16。
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

参数配置

成员名称	取值范围	特殊说明
transposeA	false/true	取值为true时，不支持部分场景，详见**规格说明**。
transposeB	false/true	受硬件约束。
hasBias	false/true	受硬件约束。
outDataType	ACL_FLOAT16/ ACL_BF16	受硬件约束。该参数对应输出tensor的数据类型。
enAccum	false	-
matmulType	MATMUL_UNDEFINED	-

输入

参数	维度	数据类型	格式	描述
x	[m, k]/[batch, m, k]	int8	ND	矩阵乘的A矩阵。
weight	[k, n]/[batch, k, n]/[1, n / 16, k, 16]/[batch, n / 16, k, 16]	int8	ND/NZ	矩阵乘的B矩阵，权重。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
bias	[1, n]/[n]/[batch, n]	int32	ND	叠加的偏置矩阵。当hasBias取值为true时输入。
deqScale	[1, n]/[n]/[batch, n]	int64/uint64/float	ND	反量化步长。

输出

参数	维度	数据类型	格式	描述
output	[m, n]/[batch, m, n]	float16/bf16	ND	矩阵乘反量化计算结果。

规格说明

由于输入输出的排列组合约束较复杂，下表列举了所有输入输出属性的组合，表格中没有的组合即不支持：

图 10-43 输入输出属性排列组合

	x			weight			bias			deqScale			output		
组合ID	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度	数据类型	格式	维度
1	int8	ND	[m, k]	int8	ND	[k, n]	int32	ND	[1, n]/[n]	int64/uint64	ND	[1, n]/[n]	float16	ND	[m, n]
2	int8	ND	[batch, m, k]	int8	ND	[k, n]	int32	ND	[1, n]/[n]	int64/uint64	ND	[1, n]/[n]	float16	ND	[batch, m, n]
3	int8	ND	[batch, m, k]	int8	ND	[batch, k, n]	int32	ND	[batch, n]	int64/uint64	ND	[batch, n]	float16	ND	[batch, m, n]
4	int8	ND	[m, k]	int8	NZ	[k, n]/[1, n / 32, k, 32]	int32	ND	[1, n]/[n]	int64/uint64	ND	[1, n]/[n]	float16	ND	[m, n]
5	int8	ND	[batch, m, k]	int8	NZ	[k, n]/[1, n / 32, k, 32]	int32	ND	[1, n]/[n]	int64/uint64	ND	[1, n]/[n]	float16	ND	[batch, m, n]
6	int8	ND	[batch, m, k]	int8	NZ	[batch, k, n]/[batch, n / 32, k, 32]	int32	ND	[batch, n]	int64/uint64	ND	[batch, n]	float16	ND	[batch, m, n]
7	int8	ND	[m, k]	int8	ND	[k, n]	int32	ND	[1, n]/[n]	float	ND	[1, n]/[n]	bf16	ND	[m, n]
8	int8	ND	[batch, m, k]	int8	ND	[k, n]	int32	ND	[1, n]/[n]	float	ND	[1, n]/[n]	bf16	ND	[batch, m, n]
9	int8	ND	[batch, m, k]	int8	ND	[batch, k, n]	int32	ND	[batch, n]	float	ND	[batch, n]	bf16	ND	[batch, m, n]
10	int8	ND	[m, k]	int8	ND	[k, n]	int32	ND	[1, n]/[n]	float	ND	[1, n]/[n]	float16	ND	[m, n]
11	int8	ND	[batch, m, k]	int8	ND	[k, n]	int32	ND	[1, n]/[n]	float	ND	[1, n]/[n]	float16	ND	[batch, m, n]
12	int8	ND	[batch, m, k]	int8	ND	[batch, k, n]	int32	ND	[batch, n]	float	ND	[batch, n]	float16	ND	[batch, m, n]
13	int8	ND	[m, k]	int8	NZ	[k, n]/[1, n / 32, k, 32]	int32	ND	[1, n]/[n]	float	ND	[1, n]/[n]	float16	ND	[m, n]
14	int8	ND	[batch, m, k]	int8	NZ	[k, n]/[1, n / 32, k, 32]	int32	ND	[1, n]/[n]	float	ND	[1, n]/[n]	float16	ND	[batch, m, n]
15	int8	ND	[batch, m, k]	int8	NZ	[batch, k, n]/[batch, n / 32, k, 32]	int32	ND	[batch, n]	float	ND	[batch, n]	float16	ND	[batch, m, n]

- Atlas 推理系列产品不支持组合7-15。

 版权所有 © 华为技术有限公司

- Atlas 800I A2 推理产品/Atlas A2 训练系列产品不支持组合10-15。
- transposeA取值为true时，不支持组合2、5、8、11、14。

- 当weight维度为4维时，若“transposeB”为false，n为32的整数倍，k为16的整数倍；若“transposeB”为true，k为32的整数倍，n为16的整数倍。
- Atlas 训练系列产品不支持组合1-9。

OP 使用与典型场景

OP使用时，可参考算子使用指导中的使用流程部分，其中，单算子（OpsOperation）构造Operation参数的构造方法参考以下参数构造部分。

```
// 参数构造
atb::infer::LinearParam param;
param.transposeA = false;
param.transposeB = false;
param.hasBias = true;
param.outDataType = ACL_FLOAT16;
param.enAccum = false;
param.matmulType = MATMUL_UNDEFINED;
# 计算示例
>>> x
tensor([[1, 2],
        [3, 4]])
>>> weight
tensor([[1, 2, 3],
        [4, 5, 6]])
>>> bias
tensor([1, 2, 3])
>>> deqScale
tensor([1, 2, 3])
>>> output
tensor([[10, 28, 54],
        [20, 56, 108]])
# 10 = (1 * 1 + 2 * 4 + 1) * 1
# 28 = (1 * 2 + 2 * 5 + 2) * 2
# 54 = (1 * 3 + 2 * 6 + 3) * 3
# 20 = (3 * 1 + 4 * 4 + 1) * 1
# 56 = (3 * 2 + 4 * 5 + 2) * 2
# 108 = (3 * 3 + 4 * 6 + 3) * 3
```

10.35.6.5 matmul 使能爱因斯坦乘

功能概述

矩阵乘matmul使能爱因斯坦乘。

计算公式

矩阵乘输入两个张量 $A_{mbk}(x)$ 和 $B_{bkn}(weight)$ ，输出张量为 C_{mbn} ：

$$C_{mbn} = A_{mbk} * B_{bkn}$$

 版权所有 © 华为技术有限公司

硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 推理系列产品	不支持	-
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

参数配置

成员名称	取值范围	特殊说明
transposeA	false	-
transposeB	false/true	-
hasBias	false	-
outDataType	ACL_DT_UNDEFINED	-
enAccum	false	-
matmulType	MATMUL_EIN_SUM	-

输入

参数	维度	数据类型	格式	描述
x	[m, batch, k]	float16	ND	矩阵乘的A矩阵。
weight	[batch, k, n]	float16	ND	矩阵乘的B矩阵，权重。

输出

参数	维度	数据类型	格式	描述
output	[m, batch, n]	float16	ND	矩阵乘计算结果。

OP 使用与典型场景

OP使用时，可参考算子使用指导中的使用流程部分，其中，单算子（OpsOperation）构造Operation参数的构造方法参考下列各场景的参数构造部分。

```
// 参数构造
atb::infer::LinearParam param;
param.transposeA = false;
```

 版权所有 © 华为技术有限公司

```
param.transposeB = false;
param.hasBias = false;
param.outDataType = ACL_DT_UNDEFINED;
param.enAccum = false;
param.matmulType = MATMUL_EIN_SUM;
# 计算示例
>>> x
tensor([[[1, 2]],
        [[3, 4]]])
>>> weight
tensor([[[1, 2, 3],
         [4, 5, 6]]])
>>> output
tensor([[[9, 12, 15],
       [19, 26, 33]]])
# 9 = 1 * 1 + 2 * 4
# 12 = 1 * 2 + 2 * 5
# 15 = 1 * 3 + 2 * 6
# 19 = 3 * 1 + 4 * 4
# 26 = 3 * 2 + 4 * 5
# 33 = 3 * 3 + 4 * 6
```

10.35.7 算子调用示例（C++）

10.35.7.1 linear_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include <atb/atb_infer.h>

#include "demo_util.h"
const int32_t DEVICE_ID = 0;
const uint32_t X_DIM_0 = 2;
const uint32_t X_DIM_1 = 3;
const uint32_t WEIGHT_DIM_0 = 3;
const uint32_t WEIGHT_DIM_1 = 2;
const uint32_t BIAS_DIM_0 = 2;

/**
 * @brief 准备atb::VariantPack
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建shape为[2, 3]的输入x tensor
    atb::Tensor xFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>{1, 2, 3, 4, 5, 6},
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {X_DIM_0, X_DIM_1});
    // 创建shape为[3, 2]的输入weight tensor
    atb::Tensor weightFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>{1, 2, 3, 4, 5, 6},
        ACL_FLOAT16,
```

 版权所有 © 华为技术有限公司

```
        aclFormat::ACL_FORMAT_ND,
        {WEIGHT_DIM_0, WEIGHT_DIM_1});
    // 创建shape为[2]的输入bias tensor
    atb::Tensor biasFloat = CreateTensorFromVector(
        contextPtr, stream, std::vector<float>(BIAS_DIM_0, 1.0), ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{1, BIAS_DIM_0});
    atb::SVector<atb::Tensor> inTensors = {xFloat, weightFloat, biasFloat};
    return inTensors;
}

/**
 * @brief 创建一个linear operation
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *CreateLinearOperation()
{
    atb::infer::LinearParam param;
    param.transposeA = false;
    param.transposeB = false;
    param.hasBias = true;
    param.outDataType = aclDataType::ACL_DT_UNDEFINED;
    param.enAccum = false;
    param.matmulType = atb::infer::LinearParam::MATMUL_UNDEFINED;
    atb::Operation *LinearOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &LinearOp));
    return LinearOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    atb::Context *context = nullptr;
    void *stream = nullptr;

    CHECK_STATUS(aclInit(nullptr));
    CHECK_STATUS(aclrtSetDevice(DEVICE_ID));
    CHECK_STATUS(atb::CreateContext(&context));
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 创建op
    atb::Operation *linearOp = CreateLinearOperation();
    // 准备输入tensor
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    // 准备输出tensor
    atb::Tensor output = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {X_DIM_0,
WEIGHT_DIM_1});
    variantPack.outTensors = {output};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(linearOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // linear执行
    linearOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放资源
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
```

 版权所有 © 华为技术有限公司

```
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(linearOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    std::cout << "Linear demo success!" << std::endl;
    return 0;
}
```

10.35.7.2 linear_einsum_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：爱因斯坦乘场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include <atb/atb_infer.h>

#include "demo_util.h"
const int32_t DEVICE_ID = 0;
const uint32_t XDIM_0 = 2;
const uint32_t XDIM_1 = 3;
const uint32_t WEIGHTDIM_0 = 3;
const uint32_t WEIGHTDIM_1 = 2;
const uint32_t BIASDIM_0 = 2;
const uint32_t BATCHSIZE = 1;

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建shape为[2, 3]的输入x tensor
    atb::Tensor xFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>{1, 2, 3, 4, 5, 6},
        aclDataType::ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {XDIM_0, BATCHSIZE, XDIM_1});
    // 创建shape为[3, 2]的输入weight tensor
    atb::Tensor weightFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>{1, 2, 3, 4, 5, 6},
        aclDataType::ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {BATCHSIZE, WEIGHTDIM_0, WEIGHTDIM_1});
    atb::SVector<atb::Tensor> inTensors = {xFloat, weightFloat};
    return inTensors;
}


/**
 * @brief 创建一个linear operation
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *CreateLinearOperation()
```

 版权所有 © 华为技术有限公司

```
{
    atb::infer::LinearParam param;
    param.transposeA = false;
    param.transposeB = false;
    param.hasBias = false;
    param.outDataType = aclDataType::ACL_DT_UNDEFINED;
    param.enAccum = false;
    param.matmulType = atb::infer::LinearParam::MATMUL_EIN_SUM;
    atb::Operation *LinearOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &LinearOp));
    return LinearOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    atb::Context *context = nullptr;
    void *stream = nullptr;

    CHECK_STATUS(aclInit(nullptr));
    CHECK_STATUS(aclrtSetDevice(DEVICE_ID));
    CHECK_STATUS(atb::CreateContext(&context));
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 创建op
    atb::Operation *linearOp = CreateLinearOperation();
    // 准备输入tensor
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    // 准备输出tensor
    atb::Tensor output = CreateTensor(aclDataType::ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {XDIM_0,
BATCHSIZE, WEIGHTDIM_1});
    variantPack.outTensors = {output};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(linearOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // linear执行
    linearOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放资源
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(linearOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    std::cout << "Linear demo success!" << std::endl;
    return 0;
}
```

 版权所有 © 华为技术有限公司

10.35.7.3 linear_dequant_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：量化场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include <atb/atb_infer.h>

#include "demo_util.h"
const int32_t DEVICE_ID = 0;
const uint32_t XDIM_0 = 2;
const uint32_t XDIM_1 = 3;
const uint32_t WEIGHTDIM_0 = 3;
const uint32_t WEIGHTDIM_1 = 2;

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建shape为[2, 3]的输入x tensor
    atb::Tensor x = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<int8_t>{1, 2, 3, 4, 5, 6},
        aclDataType::ACL_INT8,
        aclFormat::ACL_FORMAT_ND,
        {XDIM_0, XDIM_1});
    // 创建shape为[3, 2]的输入weight tensor
    atb::Tensor weight = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<int8_t>{1, 2, 3, 4, 5, 6},
        aclDataType::ACL_INT8,
        aclFormat::ACL_FORMAT_ND,
        {WEIGHTDIM_0, WEIGHTDIM_1});
    // 创建shape为[2]bias tensor
    atb::Tensor bias = CreateTensorFromVector(
        contextPtr, stream, std::vector<int32_t>(2, 1), aclDataType::ACL_INT32, aclFormat::ACL_FORMAT_ND,
{1, 2});
    // 创建shape为[2]的输入deqScale tensor
    atb::Tensor deqScale = CreateTensorFromVector(
        contextPtr, stream, std::vector<float>(2, 1), aclDataType::ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {1,
2});
    atb::SVector<atb::Tensor> inTensors = {x, weight, bias, deqScale};
    return inTensors;
}

/**
 * @brief 创建一个linear operation
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *CreateLinearOperation()
{
    atb::infer::LinearParam param;
    param.transposeA = false;
    param.transposeB = false;
    param.hasBias = true;
    param.outDataType = aclDataType::ACL_BF16;
    param.enAccum = false;
```

 版权所有 © 华为技术有限公司

```
    atb::Operation *LinearOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &LinearOp));
    return LinearOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    atb::Context *context = nullptr;
    void *stream = nullptr;

    CHECK_STATUS(aclInit(nullptr));
    CHECK_STATUS(aclrtSetDevice(DEVICE_ID));
    CHECK_STATUS(atb::CreateContext(&context));
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 创建op
    atb::Operation *linearOp = CreateLinearOperation();
    // 准备输入tensor
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    // 准备输出tensor
    atb::Tensor output = CreateTensor(aclDataType::ACL_BF16, aclFormat::ACL_FORMAT_ND, {XDIM_0,
WEIGHTDIM_1});
    variantPack.outTensors = {output};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(linearOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // linear执行
    linearOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放资源
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(linearOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    std::cout << "Linear dequant demo success!" << std::endl;
    return 0;
}
```

10.35.7.4 linear_dequant_inference_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 推理系列产品。

场景：量化场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
```

```
#include <atb/atb_infer.h>

#include "demo_util.h"
const int32_t DEVICE_ID = 0;
const uint32_t X_DIM_0 = 2;
const uint32_t X_DIM_1 = 3;
const uint32_t WEIGHT_DIM_0 = 3;
const uint32_t WEIGHT_DIM_1 = 2;
const uint32_t BIAS_DIM_0 = 2;

/**
 * @brief 准备atb::VariantPack
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建shape为[2, 3]的输入x tensor
    atb::Tensor xFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>{1, 2, 3, 4, 5, 6},
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {X_DIM_0, X_DIM_1});
    // 创建shape为[3, 2]的输入weight tensor
    atb::Tensor weightFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>{1, 2, 3, 4, 5, 6},
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {WEIGHT_DIM_0, WEIGHT_DIM_1});
    // 创建shape为[2]的输入bias tensor
    atb::Tensor biasFloat = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>(BIAS_DIM_0, 1.0),
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {1, BIAS_DIM_0});
    atb::SVector<atb::Tensor> inTensors = {xFloat, weightFloat, biasFloat};
    return inTensors;
}

/**
 * @brief 创建一个linear operation
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *CreateLinearOperation()
{
    atb::infer::LinearParam param;
    param.transposeA = false;
    param.transposeB = false;
    param.hasBias = true;
    param.outDataType = aclDataType::ACL_DT_UNDEFINED;
    param.enAccum = false;
    param.matmulType = atb::infer::LinearParam::MATMUL_UNDEFINED;
    atb::Operation *LinearOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &LinearOp));
    return LinearOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    atb::Context *context = nullptr;
    void *stream = nullptr;

    CHECK_STATUS(aclInit(nullptr));
```

 版权所有 © 华为技术有限公司

```
    CHECK_STATUS(aclrtSetDevice(DEVICE_ID));
    CHECK_STATUS(atb::CreateContext(&context));
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 创建op
    atb::Operation *linearOp = CreateLinearOperation();
    // 准备输入tensor
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    // 准备输出tensor
    atb::Tensor output = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {X_DIM_0,
WEIGHT_DIM_1});
    variantPack.outTensors = {output};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(linearOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // linear执行
    linearOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放资源
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(linearOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    std::cout << "Linear dequant demo success!" << std::endl;
    return 0;
}
```

10.36 LinearParallelOperation

10.36.1 功能描述

通信计算并行算子，该算子功能为linear和通信算子组合，通信和计算是并行处理，与串行处理相较性能大幅度提升。

该算子涉及多卡相关操作，可根据实际需求配置HCCL相关环境变量，具体请参见《CANN 环境变量参考》中的“**集合通信**”章节。

10.36.2 定义

```
struct LinearParallelParam {
    enum ParallelType : int {
        UNDEFINED = -1,
        LINEAR_ALL_REDUCE = 0,
        LINEAR_REDUCE_SCATTER = 1,
        ALL_GATHER_LINEAR = 2,
        PURE_LINEAR = 3,
```

 版权所有 © 华为技术有限公司

```
        ALL_GATHER_LINEAR_REDUCE_SCATTER = 4,
        MAX = 5,
    };
    enum QuantType : int {
        QUANT_TYPE_UNDEFINED = -1,
        QUANT_TYPE_UNQUANT = -1,
        QUANT_TYPE_PER_TENSOR = 0,
        QUANT_TYPE_PER_CHANNEL = 1,
        QUANT_TYPE_PER_GROUP = 2,
        QUANT_TYPE_MAX = 3,
    };
    bool transWeight = true;
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    bool hasResidual = false;
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    ParallelType type = LINEAR_ALL_REDUCE;
    bool keepIntermediate = false;
    QuantType quantType = QUANT_TYPE_UNDEFINED;
    int32_t quantGroupSize = 0;
    aclDataType outDataType = ACL_DT_UNDEFINED;
    std::string commDomain;
    struct TwoDimTPInfo {
        uint16_t agDim = 0;
        uint16_t rsDim = 0;
        uint8_t innerDimIsAg = 1;
        uint8_t rsv[3] = {0};
    };
    TwoDimTPInfo twoDimTPInfo;
    uint8_t rsv[56] = {0};
};
```

10.36.3 参数列表

成员名称	类型	默认值	描述
transWeight	bool	true	权重是否需要转置，默认为true。
rank	int	0	当前卡所属通信编号。
rankSize	int	0	通信的卡的数量。
rankRoot	int	0	主通信编号。
hasResidual	bool	false	是否叠加残差。配置为“false”时不叠加残差，为“true”时叠加残差。默认不叠加残差。
backend	std::string	"hccl"	通信后端指示。支持“hccl”，“lccl”，“lcoc”。 Atlas 推理系列产品仅支持“backend”为“hccl”。
hcclComm	HcclComm	nullptr	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
commMode	CommMode	COMM_MULTI_PROCESS	通信模式，**CommMode**类型枚举值。hccl多线程只支持外部传入通信域方式。
rankTableFile	std::string	-	集群信息的配置文件路径，适用单机以及多机通信场景，当前仅支持hccl后端场景。 配置请参见《 TensorFlow 1.15模型迁移指南 》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。
type	ParallelType	LINEAR_ALL_REDUCE	权重并行类型。仅在“backend”为“lcoc”时生效。 • UNDEFINED：未定义。 • LINEAR_ALL_REDUCE：linear + AllReduce，默认值。 • LINEAR_REDUCE_SCATTER：linear + reduce_scatter。 • ALL_GATHER_LINEAR：AllGather + linear。 • PURE_LINEAR：linear。 • ALL_GATHER_LINEAR_REDUCE_SCATTER：AllGather + linear + reduce_scatter。 • MAX ： 枚举类型最大值。
keepIntermediate	bool	false	是否返回中间结果，仅在“ParallelType”使用“ALL_GATHER_LINEAR”时生效。
quantType	QuantType	QUANT_TYPE_UNQUANT	量化类型。仅在“backend”为“lcoc”时生效。 • QUANT_TYPE_UNDEFINED：默认值。 • QUANT_TYPE_UNQUANT：默认值。 • QUANT_TYPE_PER_TENSOR：对整个张量进行量化。 • QUANT_TYPE_PER_CHANNEL：对张量中每个channel分别进行量化。 • QUANT_TYPE_PER_GROUP：将张量按quantGroupSize划分后，分别进行量化。 • QUANT_TYPE_MAX：枚举类型最大值。
quantGroupSize	int32_t	0	量化类型为“QUANT_TYPE_PER_GROUP”时有效。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
outDataType	aclDataType	ACL_DT_UNDEFINED	• 若为浮点linear，参数“outDataType”配置为ACL_DT_UNDEFINED，表示输出tensor的数据类型与输入tensor一致。 • 若为量化linear，输出tensor的数据类型与输入tensor不一致，则参数“outDataType”配置为用户预期输出tensor的数据类型, 如ACL_FLOAT16/ACL_BF16。
commDomain	std::string	-	通信device组用通信域名标识，多通信域时使用。当backend为lccl时，commMode为多进程时，commDomain需要设置0-63。
twoDimTPInfo	TwoDimTPInfo	-	AllGather_Matmul_ReduceScatter算子参数。
rsv[56]	uint8_t	{0}	预留参数。

表 10-7 TwoDimTPInfo 成员

成员名称	类型	默认值	描述
agDim	uint16_t	0	表示allGather轴卡数，规定x轴方向是非连续卡号。
rsDim	uint16_t	0	表示reduceScatter轴卡数，规定y轴方向是连续卡号。
innerDimIsAg	uint8_t	1	allGather通信的rank是否连续，1表示true，0表示false。
rsv[3]	uint8_t	{0}	填充满8字节。

10.36.4 输入输出

输入

参数	维度	数据类型	格式	描述
input	[m, k]/[batch, m, k]	• 浮点：float16/bf16 • 量化：float16/bf16/int8	ND	矩阵乘运算的A矩阵。 k为32的整数倍。 当type为ALL_GATHER_LINEAR_REDUCE_SCATTER时，不支持量化。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
weight	[k, n] NZ：浮点额外支持[1, n/16, k, 16]	• 浮点：float16/bf16 • 量化：int8	• 浮点：ND/NZ • 量化：ND	权重，矩阵乘的B矩阵。“backend”为“lcoc”时float16/bf16支持NZ；“backend”为“hccl”或“lccl”时仅float16支持NZ。 当type为ALL_GATHER_LINEAR_REDUCE_SCATTER时，不支持量化。
bias	• “quantType”为“per_tensor”时支持：[1] • “quantType”为“per_channel”时支持：[1, n]/[n] • “quantType”“为”“per_group”时支持：[k/quantGroupSize, n]	• W8A16量化场景：float16/bf16 • W8A8量化场景：int32	ND	叠加的偏置矩阵。 n为16的整数倍。 ALL_GATHER_LINEAR_REDUCE_SCATTER时，不支持。
deqScale	• “quantType”为“per_tensor”时支持：[1] • “quantType”为“per_channel”时支持：[1, n]/[n] • “quantType”“为”“per_group”时支持：[k/quantGroupSize, n]	• W8A16量化场景：float16/bf16 • W8A8量化场景：int64	ND	反量化的scale。 量化时输入。 当type为ALL_GATHER_LINEAR_REDUCE_SCATTER时不支持。
residual	[n]	float16/bf16	ND	残差，用于叠加到最后的输出结果上。 当type为ALL_GATHER_LINEAR_REDUCE_SCATTER时不支持。

 版权所有 © 华为技术有限公司

输出

参数	维度	数据类型	格式	描述
output	• 当type为linear_all_reduce/pure_linear：[m, n]/[batch, m, n] • 当type为linear_reduce_scatter：[m/rankSize, n]/[batch/rankSize, m, n] • 当type为all_gather_linear：[m*rankSize, n]/[batch*rankSize, m, n]; • 当type为ALL_GATHER_LINEAR_REDUCE_SCATTER：[m*agDim/rsDim, n]	float16/bf16	ND	输出tensor，维度数与x一致。
intermediateOutput	[m*rankSize, n]/[batch*rankSize, m, n]	float16/bf16	ND	输出tensor，维度数与x一致。 “keepIntermediate”为true且“ParallelType”使用“ALL_GATHER_LINEAR”时生效。

10.36.5 规格约束

- 输入x / weight矩阵维度，通过transWeight配置需满足矩阵乘的维度关系。
- rank、rankSize、rankRoot需满足以下条件。
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：

```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```

- 调用AllGatherMatmulReduceScatter算子时（即LinearParallelParam的type设置为ALL_GATHER_LINEAR_REDUCE_SCATTER时），backend需要设置为"lcoc"，

 版权所有 © 华为技术有限公司

keepIntermediate需为false，twoDimTPInfo中的agDim、rsDim、innerDimIsAg可供设置，且需满足agDim * rsDim == rankSize，不支持量化相关参数。输入维度m、k、n的取值范围分别为[1, 200000]、[1, 32768]、[1, 32768]。

- agDim和rsDim需为2的幂，且最小为2，rankSize最小为4，最大为16。
- Atlas 200T A2 Box16 异构子框场景下，通信域仅支持前8卡或后8卡内部建立，不支持同时包含前8卡和后8卡的通信域。

10.36.6 算子调用示例（C++）

10.36.6.1 linear_parallel_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <acl/acl.h>
#include <atb/atb_infer.h>
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>

#include "demo_util.h"

void ExcuteImpl(atb::Operation *op, atb::VariantPack variantPack, atb::Context *context)
{
    uint64_t workspaceSize = 0;
    CHECK_STATUS(op->Setup(variantPack, workspaceSize, context));
    void *workspace = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc(&workspace, workspaceSize, ACL_MEM_MALLOC_HUGE_FIRST));
    }
    CHECK_STATUS(op->Execute(variantPack, (uint8_t *)workspace, workspaceSize, context));

    if (workspace) {
        CHECK_STATUS(aclrtFree(workspace));  // 销毁workspace
    }
}
void LinearParallelSample(int rank, int rankSize)
{
    int ret = aclInit(nullptr);
    // 设置每个进程对应的deviceId
    int deviceId = rank;
    CHECK_STATUS(aclrtSetDevice(deviceId));

    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    aclrtStream stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    atb::Tensor input = CreateTensorFromVector(
        context, stream, std::vector<float>(64, 2.0), aclDataType::ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{2, 32});
    atb::Tensor weight = CreateTensorFromVector(
        context, stream, std::vector<float>(64, 2.0), aclDataType::ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{32, 2});

    atb::Tensor output;
    output.desc.dtype = ACL_FLOAT16;
    output.desc.format = ACL_FORMAT_ND;
    output.desc.shape.dimNum = 2;
    output.desc.shape.dims[0] = 2;
    output.desc.shape.dims[1] = 2;
    output.dataSize = atb::Utils::GetTensorSize(output);
```

 版权所有 © 华为技术有限公司

```
    CHECK_STATUS(aclrtMalloc(&output.deviceData, output.dataSize, ACL_MEM_MALLOC_HUGE_FIRST));

    atb::infer::LinearParallelParam param;
    param.transWeight = false;
    param.rank = rank;
    param.rankRoot = 0;
    param.rankSize = rankSize;
    param.backend = "hccl";
    atb::Operation *op = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &op));

    atb::VariantPack variantPack;
    variantPack.inTensors = {input, weight};
    variantPack.outTensors = {output};
    ExcuteImpl(op, variantPack, context);
    std::cout << "rank: " << rank << " executed END." << std::endl;

    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(op));     // 销毁op对象
    CHECK_STATUS(aclrtDestroyStream(stream));    // 销毁stream
    CHECK_STATUS(atb::DestroyContext(context));  // 销毁context
    CHECK_STATUS(aclFinalize());
    std::cout << "demo excute success" << std::endl;
}

int main(int argc, const char *argv[])
{
    const int processCount = 2;
    for (int i = 0; i < processCount; i++) {
        pid_t pid = fork();
        // 子进程
        if (pid == 0) {
            LinearParallelSample(i, processCount);
            return 0;
        } else if (pid < 0) {
            std::cerr << "Failed to create process." << std::endl;
            return 1;
        }
    }
    // 父进程等待子进程执行完成
    for (int i = 0; i < processCount; ++i) {
        wait(NULL);
    }
    std::cout << "The communication operator is successfully executed. Parent process exit" << std::endl;
    return 0;
}
```

10.36.6.2 linear_parallel_quant_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：量化场景。

```
#include <acl/acl.h>
#include <atb/atb_infer.h>
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>

#include "demo_util.h"

void ExcuteImpl(atb::Operation *op, atb::VariantPack variantPack, atb::Context *context)
{
    uint64_t workspaceSize = 0;
    CHECK_STATUS(op->Setup(variantPack, workspaceSize, context));
    void *workspace = nullptr;
    if (workspaceSize > 0) {
```

 版权所有 © 华为技术有限公司

```
        CHECK_STATUS(aclrtMalloc(&workspace, workspaceSize, ACL_MEM_MALLOC_HUGE_FIRST));
    }
    CHECK_STATUS(op->Execute(variantPack, (uint8_t *)workspace, workspaceSize, context));

    if (workspace) {
        CHECK_STATUS(aclrtFree(workspace));  // 销毁workspace
    }
}
void LinearParallelSample(int rank, int rankSize)
{
    int ret = aclInit(nullptr);
    // 设置每个进程对应的deviceId
    int deviceId = rank;
    CHECK_STATUS(aclrtSetDevice(deviceId));

    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    aclrtStream stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    atb::Tensor input = CreateTensorFromVector(
        context, stream, std::vector<float>(64, 2.0), aclDataType::ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{2, 32});

    atb::Tensor weight = CreateTensorFromVector(
        context, stream, std::vector<int8_t>(64, 2), aclDataType::ACL_INT8, aclFormat::ACL_FORMAT_ND, {32,
2});

    atb::Tensor bias = CreateTensorFromVector(
        context, stream, std::vector<float>(1, 1.0), aclDataType::ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{1});

    atb::Tensor deqScale = CreateTensorFromVector(
        context, stream, std::vector<float>(1, 1.0), aclDataType::ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{1});

    atb::Tensor output = CreateTensor(aclDataType::ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {2, 2});

    atb::infer::LinearParallelParam param;
    param.transWeight = false;
    param.rank = rank;
    param.rankRoot = 0;
    param.rankSize = rankSize;
    param.backend = "lcoc";
    param.keepIntermediate = false;
    param.type = atb::infer::LinearParallelParam::ParallelType::LINEAR_ALL_REDUCE;
    param.quantType = atb::infer::LinearParallelParam::QuantType::QUANT_TYPE_PER_TENSOR;
    param.outDataType = ACL_FLOAT16;
    atb::Operation *op = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &op));

    atb::VariantPack variantPack;
    variantPack.inTensors = {input, weight, bias, deqScale};
    variantPack.outTensors = {output};
    ExcuteImpl(op, variantPack, context);
    std::cout << "rank: " << rank << " executed END." << std::endl;

    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(op));     // 销毁op对象
    CHECK_STATUS(aclrtDestroyStream(stream));    // 销毁stream
    CHECK_STATUS(atb::DestroyContext(context));  // 销毁context
    CHECK_STATUS(aclFinalize());
    std::cout << "demo excute success" << std::endl;
}

int main(int argc, const char *argv[])
{
    const int processCount = 2;
```

 版权所有 © 华为技术有限公司

```
    for (int i = 0; i < processCount; i++) {
        pid_t pid = fork();
        // 子进程
        if (pid == 0) {
            LinearParallelSample(i, processCount);
            return 0;
        } else if (pid < 0) {
            std::cerr << "Failed to create process." << std::endl;
            return 1;
        }
    }
    // 父进程等待子进程执行完成
    for (int i = 0; i < processCount; ++i) {
        wait(NULL);
    }
    std::cout << "The communication operator is successfully executed. Parent process exit" << std::endl;
    return 0;
}
```

10.37 LinearSparseOperation

功能

稀疏量化Linear。

该算子实现功能与量化linear类似。不同点在于稀疏量化算子会使用压缩工具提前对weight输入进行压缩，以此提升算子性能。参数tilingK和tilingN由压缩算法决定，目前均只支持取值为8。 目前该算子仅支持在Atlas 推理系列产品中进行运算。

定义

```
struct LinearSparseParam{
    bool transposeA = false;
    bool transposeB = true;
    uint32_t tilingK =8;
    uint32_t tilingN = 8;
    uint8_t rsv[12] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
transposeA	bool	false	是否转置A矩阵。在量化场景下，非Atlas 800I A2 推理产品/Atlas A2 训练系列产品仅支持配置为false。
transposeB	bool	true	是否转置B矩阵。在反量化场景下，非Atlas 800I A2 推理产品/Atlas A2 训练系列产品仅支持配置为true。
tilingK	uint32_t	8	压缩参数，由外部压缩算法决定，目前仅支持取值为8。
tilingN	uint32_t	8	压缩参数，由外部压缩算法决定，目前仅支持取值为8。
rsv[12]	uint8_t	{0}	预留参数。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
x	[m, k]	int8	ND	矩阵乘运算的A矩阵。 m需小于等于256，k为64的整数倍且大于256.
weight	[c]	int8	ND/NZ	权重，矩阵乘的B矩阵。 通过压缩工具压缩后的权重，shape大小c的值大于0且不大于k * n。
bias	[1, n] 或 [n]	int32	ND	叠加的偏置矩阵。n为64的整数倍且大于等于128。
deqScale	[1, n] 或 [n]	uint64/int64	ND	反量化的scale。量化时输入。
compressIdx	[x]	int8	ND	压缩权重时同时生成的压缩索引，x由以下计算得出。 x = k_index * n_index * 8 k_index = ceil(k1 / tilingK) n_index = ceil(n1 / tilingN) k1 = k / 32 n1 = n / 16 其中，ceil()函数为向上取整。

输出

参数	维度	数据类型	格式	描述
output	[m, n]	float16	ND	输出tensor，维度数与x一致。

规格约束

- 输入x / weight原维度，通过transposeA / transposeB配置要满足矩阵乘的维度关系。
- 输入weight的维度c由压缩工具（如ModelSlim）计算而来，满足c大于0且小于k * n。
- 该算子仅支持在Atlas 推理系列产品中进行运算。
- 参数tilingK和tilingN由压缩算法决定，目前均只支持取值为8。

 版权所有 © 华为技术有限公司

10.38 MlaPreprocessOperation

功能

融合了MLA场景下PagedAttention输入数据处理的全过程，包括从隐状态输入开始经过rmsnorm、反量化、matmul、rope、reshapeAndCache的一系列计算。

 版权所有 © 华为技术有限公司

图 10-44 计算流程图



 版权所有 © 华为技术有限公司

硬件支持情况

硬件型号	支持情况
Atlas 推理系列产品	不支持
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持

定义

```
struct MlaPreprocessParam{
    uint32_t wdqDim = 0;
    uint32_t qRopeDim = 0;
    uint32_t kRopeDim = 0;
    float epsilon = 1e-5;
    int32_t qRotaryCoeff = 2;
    int32_t kRotaryCoeff = 2;
    bool transposeWdq = true;
    bool transposeWuq = true;
    bool transposeWuk = true;
    enum CacheMode : uint8_t {
        KVCACHE = 0,
        KROPE_CTKV,
        INT8_NZCACHE,
        NZCACHE,
    };
    CacheMode cacheMode = KVCACHE;
    enum QuantMode : uint16_t {
        PER_TENSOR_QUANT_ASYMM = 0,
        PER_TOKEN_QUANT_SYMM,
        PER_TOKEN_QUANT_ASYMM,
        UNQUANT,
    };
    QuantMode quantMode = PER_TENSOR_QUANT_ASYMM;
    uint8_t rsv[34] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
wdqDim	uint32_t	0	[0]	否	经过matmul后拆分的dim大小。
qRopeDim	uint32_t	0	[0]	否	q传入rope的dim大小。
kRopeDim	uint32_t	0	[0]	否	k传入rope的dim大小。
epsilon	float	1e-5	[1e-5]	否	加在分母上防止除0。
qRotaryCoeff	int32_t	2	2	否	q旋转系数。
kRotaryCoeff	int32_t	2	2	否	k旋转系数。
transposeWdq	bool	true	(true)	否	wdq是否转置。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
transposeWuq	bool	true	(true)	否	wuq是否转置。
transposeWuk	bool	true	(true)	否	wuk是否转置。
cacheMode	CacheMode	0	[0,3]	否	指定cache的类型。
quantMode	QuantMode	0	[0,3]	否	指定RmsNorm量化的类型。
rsv[34]	uint8_t	{0}	[0]	否	预留字段。
当前只有cacheMode生效。					

上表中类型为自定义类型的，其描述如下：

cacheMode：表示输入query和kcache的类型，其具体取值如下。

- cacheMode为0时，kcache和q均经过拼接后输出。
- cacheMode为1时，输入输出的kvCcache拆分为krope和ctkv，q拆分为qrope和qnope。
- cacheMode为2时，krope和ctkv转为NZ格式输出，ctkv和qnope经过per_head静态对称量化为int8类型。
- cacheMode为3时，krope和ctkv转为NZ格式输出。

quantMode：表示RmsNorm量化类型，其具体取值如下。

- PER_TENSOR_QUANT_ASYMM：per_tensor静态非对称量化，默认量化类型。
- PER_TOKEN_QUANT_SYMM：per_token动态对称量化，未实现。
- PER_TOKEN_QUANT_ASYMM：per_token动态非对称量化，未实现。
- UNQUANT：不量化，浮点输出，未实现。

输入

所属模块功能	参数	维度	数据类型	格式	描述
rmsNormQuant_0	input	[tokenNum, 7168]	float16/bf16	ND	必选。
	gamma0	[7168]	float16/bf16	ND	必选。数据类型与input一致。
	beta0	[7168]	float16/bf16	ND	必选。数据类型与input一致。

 版权所有 © 华为技术有限公司

所属模块功能	参数	维度	数据类型	格式	描述
	quantScale0	[1]	float16/bf16	ND	必选，支持传入空tensor。仅在quantMode为0时传入，数据类型与input一致。
	quantOffset0	[1]	int8	ND	必选，支持传入空tensor。仅在quantMode为0时传入。
matmul_0	wdqkv	[2112,7168]	int8	NZ	必选。
	deScale0	[2112]	int64/float	ND	必选。input为float16时为int64，input为bf16时为float。
	bias0	[2112]	int32	ND	必选，支持传入空tensor。quantMode为1、3时不传入。
rmsNormQuant_1	gamma1	[1536]	float16/bf16	ND	必选。数据类型与input一致。
	beta1	[1536]	float16/bf16	ND	必选。数据类型与input一致。
	quantScale1	[1]	float16/bf16	ND	必选，支持传入空tensor。仅在quantMode为0时传入，数据类型与input一致。
	quantOffset1	[1]	int8	ND	必选，支持传入空tensor。仅在quantMode为0时传入。
matmul_1	wuq	[headNum * 192, 1536]	int8	NZ	必选。
	deScale1	[headNum * 192]	int64/float	ND	必选，input为float16时为int64，input为bf16时为float。
	bias1	[headNum * 192]	int32	ND	必选，支持传入空tensor。quantMode为1、3时不传入。

 版权所有 © 华为技术有限公司

所属模块功能	参数	维度	数据类型	格式	描述
rmsNorm	gamma2	[512]	float16/bf16	ND	必选。数据类型与input一致。
rope	cos	[tokenNum,64]	float16/bf16	ND	必选。数据类型与input一致。
	sin	[tokenNum,64]	float16/bf16	ND	必选。数据类型与input一致。
matmulEin	wuk	ND：[headNum,128,512] NZ：[headNum,32,128,16]	float16/bf16	ND/NZ	必选。数据类型与input一致。
reshapeAndCache	kvCache	• cacheMode为0：[blockNum,blockSize,1,576] • cacheMode为1：[blockNum,blockSize,1,512] • cacheMode为2：[blockNum,headNum*512/32,block_size, 32] • cacheMode为3：[blockNum,headNum*512/16,block_size, 16]	float16/bf16/int8	ND/NZ	必选。数据类型与input一致。 • cacheMode为1时，tensor的shape为拆分情况。 • cacheMode为2时格式为NZ，类型为int8。 • cacheMode为3时，格式为NZ。

 版权所有 © 华为技术有限公司

所属模块功能	参数	维度	数据类型	格式	描述
	kvCacheRope	• cacheMode为1：[blockNum, blockSize,1, 64] • cacheMode为2或3：[blockNum, headNum* 64 / 16 ,block_size, 16]	float16/ bf16	ND/ NZ	必选，支持传入空tensor。 • cacheMode不为0时传入，数据类型与input一致。 • cacheMode为2或3时，格式为NZ。
	slotmapping	[tokenNum]	int32	ND	必选。
quant	ctkvScale	[1]	float16/ bf16	ND	必选，支持传入空tensor。cacheMode为2时传入，数据类型与input一致。
	qNopeScale	[headNum]	float16/ bf16	ND	必选，支持传入空tensor。cacheMode为2时传入，数据类型与input一致。

输出

可选输出tensor不能使用空tensor占位。

所属模块功能	参数	维度	数据类型	格式	描述
输出数据	qOut0	• cacheMode为0：[tokenNum, headNum,576] • cacheMode为1或2：[tokenNum, headNum,512]	float16/ bf16/int8	ND	输出tensor。数据类型与input一致。 cacheMode为2时数据类型为int8。

 版权所有 © 华为技术有限公司

所属模块功能	参数	维度	数据类型	格式	描述
	kvCacheOut0	• cacheMode为0：[blockNum, blockSize,1, 576] • cacheMode为1：[blockNum, blockSize,1, 512] • cacheMode为2：[blockNum, headNum*512/32,block_size, 32] • cacheMode为3：[blockNum, headNum*512/16,block_size, 16]	float16/ bf16/int8	ND/ NZ	输出tensor。数据类型与input一致。 • cacheMode为2时数据类型为int8，格式为NZ。 • cacheMode为3时，格式为NZ。
	qOut1	[tokenNum,headNum,64]	float16/ bf16	ND	cacheMode不为0时输出此tensor。数据类型与input一致。
	kvCacheOut1	• cacheMode为1：[blockNum, blockSize,1, 64] • cacheMode为2：[blockNum, headNum*64 / 16 ,block_size, 16]	float16/ bf16	ND/ NZ	cacheMode不为0时输出此tensor。数据类型与input一致。 • cacheMode为2时数据格式为NZ。 • cacheMode为3时，格式为NZ。

规格约束

- tokenNum＜=1024。
- blockSize <= 128 或 blockSize = 256。
- cacheMode为2或3时，blockSize = 128。

 版权所有 © 华为技术有限公司

10.39 MultiLatentAttentionOperation

功能

MLA场景下的paged attention，使用分页管理的kvcache计算attention score，额外支持分离qnope/qrope、ctkv/krope的输入。

定义

```
struct MultiLatentAttentionParam {
    int32_t headNum = 0;
    float qkScale = 1.0;
    int32_t kvHeadNum = 0;
    enum MaskType : int {
        UNDEFINED = 0,
        MASK_TYPE_SPEC,
        MASK_TYPE_MASK_FREE,
    };
    MaskType maskType = UNDEFINED;
    enum CalcType : int {
        CALC_TYPE_UNDEFINED = 0,
        CALC_TYPE_SPEC,
        CALC_TYPE_RING,
    };
    CalcType calcType = CALC_TYPE_UNDEFINED;
    enum CacheMode : uint8_t {
        KVCACHE = 0,
        KROPE_CTKV,
        INT8_NZCACHE,
        NZCACHE,
    };
    CacheMode cacheMode = KVCACHE;
    uint8_t rsv[43] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
headNum	int32_t	0	{8,16,32,64,128}	是	query头数量。
qkScale	float	1.0	(0,1]	是	Q*K^T后乘以的缩放系数
kvHeadNum	int32_t	0	[1]	是	kv头数量
maskType	MaskType	UNDEFINED	[0,2]	否	mask类型。
calcType	CalcType	CALC_TYPE_UNDEFINED	[0,2]	否	计算类型。
cacheMode	CacheMode	KVCACHE	[0,3]	是	输入query和kcache的类型。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
rsv[43]	uint8_t	0	[0]	否	预留字段

上表中类型为自定义类型的，其描述如下：

- **maskType**：表示mask类型，其具体取值如下。
  - UNDEFINED：无mask。
  - MASK_TYPE_SPEC：并行解码mask，与calcType = CALC_TYPE_SPEC一起使用。
  - MASK_TYPE_MASK_FREE：传入固定shape的mask，与calcType = CALC_TYPE_SPEC一起使用。计算时根据尾部长度从传入mask中获取真实的mask。mask为倒三角形式，第一行全部为-inf，最后一行全部为0。具体形状如下：

```
def generate_mask_free(q_len):
    :param q_len: q_len
    :return: constructed mask, e.g. when q_len=2, returned as
    [[-10000.0 -10000.0 -10000.0 ... -10000.0],
     [0        -10000.0 -10000.0 ... -10000.0],
     [0              0 -10000.0 ... -10000.0],
     ...
     [0              0        0 ... -10000.0],
     [0              0        0 ...        0]]
    """
    mask_free = np.full((125 + 2 * q_len, 128), -10000.0)
    mask_free = np.triu(mask_free, 2 - q_len)
    return mask_free
```

- **calcType**：表示计算类型，其具体取值如下。
  - CALC_TYPE_UNDEFINED：默认的decoder场景。
  - CALC_TYPE_SPEC：支持qseqlen大于1，并行解码功能，MTP使用该场景。
  - CALC_TYPE_RING：ringAttention，额外输出lse。
- **cacheMode**：表示输入query和kcache的类型，其具体取值如下。

  与MLAPO大融合预处理算子对应使用。
  - KVCACHE：输入只有一个q和一个合并kvCache，为原PagedAttention下的MLA场景，目前暂未接入。
  - KROPE_CTKV：输入的q拆分为qNope和qRope，输入的kcache拆分为ctKV和kRope，对应之前的默认场景。
  - INT8_NZCACHE：高性能cache，在KROPE_CTKV的基础上：krope和ctkv转为NZ格式输出，ctkv和qnope经过per_head静态对称量化为int8类型。
  - NZCACHE：在KROPE_CTKV的基础上krope和ctkv转为NZ格式输出。

输入

参数	维度	数据类型	格式	cpu/npu	描述
qNope	[num_tokens, num_heads, 512]	float16/bf16/int8	ND	NPU	无位置编码query。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述
qRope	[num_tokens, num_heads, 64]	float16/bf16	ND	NPU	旋转位置编码query。
ctKV	[num_blocks , block_size, kv_heads, 512] • cacheMode为2：[blockNum, kv_heads*512/32,block_size, 32] • cacheMode为3：[blockNum, kv_heads*512/16,block_size, 16]	float16/bf16/int8	ND/NZ	NPU	无位置编码ctkv。 • cacheMode为2：int8，NZ。 • cacheMode为3：float16/bf16，NZ。
kRope	[num_blocks , block_size, kv_heads, 64] cacheMode为2或3： [blockNum, kv_heads*64 / 16 ,block_size, 16]	float16/bf16	ND/NZ	NPU	旋转位置编码k。 cacheMode为2或3：NZ。
block_tables	[batch, max_num_blocks_per_query]	int32	ND	NPU	每个query的kvcache的block映射表。
contextLens	[batch]	int32	ND	CPU	每个query对应的上下文长度，kseqlen。
mask	• MASK_TYPE_SPEC: [num_tokens(合轴), max_seq_len] • MASK_TYPE_MASK_FREE: [125 + 2 * qseqlen, 128]	float16/bf16	ND	NPU	注意力掩码，maskType不为0时传入。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述
qseqlen	[batch]	int32	ND	CPU	calcType为1时传入，每个batch对应的seqLen，取值范围为[1,4]。
qkDescale	[num_heads]	float	ND	NPU	cacheMode为2时传入。
pvDescale	[num_heads]	float	ND	NPU	cacheMode为2时传入。

输出

参数	维度	数据类型	格式	cpu/npu	描述
attenOut	[num_tokens, num_heads, head_size_vo]	float16/bf16	ND	NPU	attention输出。
lse	[num_tokens, num_heads, 1]	float16/bf16	ND	NPU	lse输出，只有calcType为CALC_TYPE_RING时需要此tensor。 只支持maskType为UNDEFINED，cacheMode为KROPE_CTKV场景。

规格约束

- block_size <= 128，建议为128。
- batch <= 8192
- cacheMode为INT8_NZCACHE时，不支持num_heads = 128

10.40 MultinomialOperation

功能

采样功能。对最后一个轴进行采样，随机抽取numSamples个值，输出下标。

定义

```
struct MultinomialParam {
    uint32_t numSamples = 1;
    uint32_t randSeed = 0;
```

 版权所有 © 华为技术有限公司

```
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
numSamples	uint32_t	1	随机采样数。 numSamples应小于等于x对应的维度大小，最大为64。
randSeed	uint32_t	0	随机数种子。
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[batch, vocSize]	float16	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
output	[batch, numSamples]	int32	ND	采数数据下标。dims[1]必须等于numSamples

规格约束

用户需确保对最后一个轴进行归一化操作。

接口调用示例

- **输入**

  intensor =

  ```
  [[0.0701, 0.1021, 0.1010, 0.1005],
  [0.1101, 0.1020, 0.1017, 0.0990],
  [0.0592, 0.1006, 0.1006, 0.1002]]
  ```

  numSamples = 1

  randSeed = 0

- **输出**

  outtensor =

  ```
  [[3],
  [3],
  [3]]
  ```

 版权所有 © 华为技术有限公司

10.41 NonzeroOperation

功能

输出非零值索引。

定义

```
struct NonzeroParam {
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	int64	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
output1	[inputDimNum, inputNumel]	int64	ND	非零元素下标。
output2	[1]	int64	ND	非零元素个数。

规格约束

仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

接口调用示例

- **一维场景**

  输入

  ```
  intensor = {[0, 1, 1, 0, 1]}
  ```

  输出

  ```
  outtensor1 = {0, 1, 2, 0, 4}outtensor2 = [3]
  ```

 版权所有 © 华为技术有限公司

- **二维场景**

  输入

  intensor = {[0, 1, 1, 0, 1]}

  输出

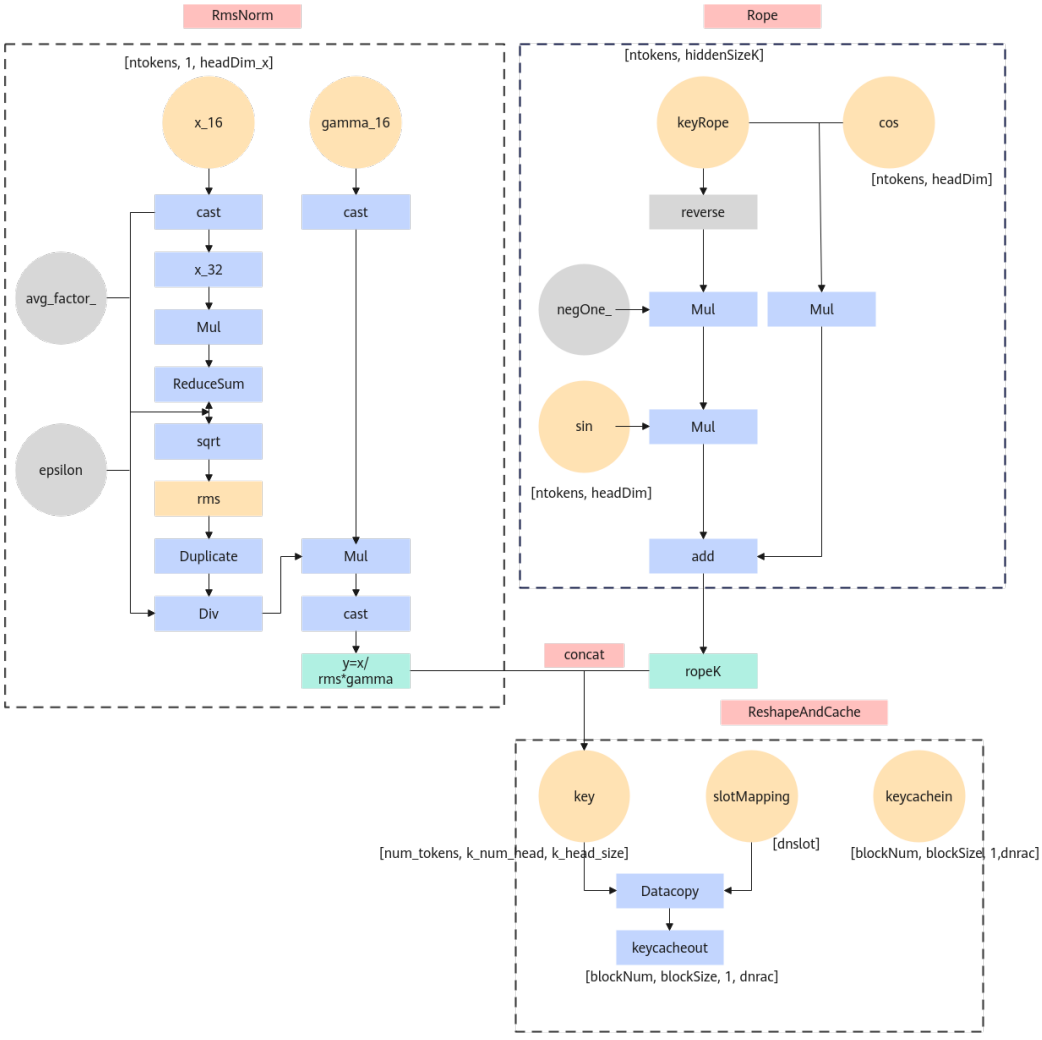
  outtensor1 = {0, 1, 2, 0, 4}outtensor2 = [3]

10.42 NormRopeReshapeOperation

功能

对输入tensor x进行RmsNorm操作，再对keyrope进行Rope操作，将两个操作的输出合并后进行ReshepeAndCache操作 。

图 10-45 计算流程图



 版权所有 © 华为技术有限公司

硬件支持情况

硬件型号	支持情况
Atlas 800I A2 推理产品	支持

定义

```
struct NormRopeReshapeParam {
    uint32_t precisionMode = 0;
    uint32_t rotaryCoeff = 2;
    float epsilon = 1e-5;
    uint8_t rsv[16] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
precisionMode	uint32_t	0	-	否	精度模式。目前只支持0。
epsilon	float	1e-5	-	否	归一化时加在分母上防止除零。
rotaryCoeff	uint32_t	2	-	否	算子内Rope部分计算的旋转系数。
rsv	uint8_t	uint8_t	-	否	预留参数。

输入

参数	维度	数据类型	格式	描述
X	[ntokens, 1, head_size_x]	float16	ND	维度目前只支持[ntokens, 1, 512]。
gamma	[head_size_x]	float16	ND	维度目前只支持[512]。
keyRope	[ntokens, hiddenSizeK]	float16	ND	[ntokens, hiddenSizeK] hiddenSizeK = headNum * head_size（这里headNum=1）。维度目前只支持[ntokens, 64]。
cos	[ntokens, head_size]	float16	ND	[ntokens, head_size]，大小与hiddenSizeK一致（headNum=1）。维度目前只支持[ntokens, 64]。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
sin	[ntokens, head_size]	float16	ND	同cos。
slotMapping	[dnslot]	int32	ND	元素值大小不能超过blockNum * blockSize。维度目前只支持[64]。
keycachein	[blockNum, blockSize, 1,dnrac]	float16	ND	dnrac=head_size_x +hiddenSizeK。维度目前只支持[blockNum, blockSize, 1, 576]。blockNum * blockSize ≥ ntokens。

输出

参数	维度	数据类型	格式	描述
keycacheout	[blockNum, blockSize, 1, dnrac]	与keycachein一致	ND	同keycachein。

规格约束

- x、gamma的最后一维大小相等且32字节对齐。
- keyRope、cos、sin的数据类型需保持一致，hiddenSizeK和head_size大小需相等。
- 11*dn*sizeof(float16) + dnrac * sizeof(float16) < maxUbSize(196352)。
- ntokens小于等于65536。

10.43 OnehotOperation

功能

onehot编码。

算子功能实现

机器学习的预处理编码方式的一种：onehot编码。onehot编码对depth个状态进行编码，每个状态的对应位置置1，其它位置置0；输出结果为只有一位为1且长为depth的向量。

算子输出输入tensor x各个元素对应的独热编码，结果表现为在axis位置上增加维度depth。

计算过程示意（Python）：

 版权所有 © 华为技术有限公司

```
res = np.eye(depth)[input0]
```

示例1（Python）：

```
x shape: torch.Size([2, 3])
x: tensor([[4, 4, 6],
          [6, 7, 6]])
depth: 10
axis: -1

# axis为-1表示output的最后一维output[i][j][:] 是x[i][j]对应的onehot编码
out_tensor shape: torch.Size([2, 3, 10])
out_tensor: tensor([[[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                     [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]],

        [[0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
         [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]]], device='npu:0')
```

示例2（Python）：

```
x shape: torch.Size([2, 3])
x: tensor([[2, 0, 4],
        [4, 3, 0]])
depth: 5
axis: 0

# output[:][i][j] 是x[i][j]对应的onehot编码，若axis为1则output[i][:][j] 是x[i][j]对应的onehot编码
out_tensor shape: torch.Size([5, 2, 3])
out_tensor: tensor([[[0, 1, 0],
         [0, 0, 1]],

        [[0, 0, 0],
         [0, 0, 0]],

        [[1, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 1, 0]],

        [[0, 0, 1],
         [1, 0, 0]]], device='npu:0')
```

定义

```
struct OnehotParam {
    int64_t axis = 0;
    int64_t depth = 0;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
axis	int64_t	0	depth所在下标。可为负数，为负数表示output的倒数第axis个维度是对应输入的独热编码。
depth	int64_t	0	每个输入对应的独热编码长度。
rsv[8]	uint8_t	{0}	预留参数。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	int32/int64	ND	输入tensor，表示要获取哪些状态的独热编码。
one	[1]	int32/int64 与x相同	ND	标量1。类型/格式与x保持一致。传给算子用，没有实际意义。
zero	[1]	int32/int64 与x相同	ND	标量0。类型/格式与x保持一致。传给算子用，没有实际意义。

输出

参数	维度	数据类型	格式	描述
output	和x相比，在axis上多一个depth维度。	int32/int64 与x相同	ND	输出tensor。类型/格式与x保持一致，输出的独热编码。

规格约束

- axis的绝对值小于x的维度数量。
- x中元素小于depth。

10.44 PadOperation

功能

对于输入input_ids，取出每个batch最后一个有效token的embedding向量。

使用场景

经过pad（末尾填充0后）之后，input_ids中有batch个token id 序列，输出时，从每个batch中拿出最后一个token的词向量，并且拼接在一起。

使用示例：

tmp_out [token_num,hidden_dim] 为[37,16]

padding_offset[1,token_num] 为[37]

seqlen[batch,1] 为[[8],[13],[16]]

则input_ids[3,16]为：

```
[[1,2,3,4,5,6,7,8,0,0,0,0,0,0,0,0],[1,2,3,4,5,6,7,8,9,10,11,12,13,0,0,0],[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]]
```

 版权所有 © 华为技术有限公司

max_seq_len=16

则output[3,16]为：

```
[[1,2,3,4,5,6,7,8,0,0,0,0,0,0,0,0],[1,2,3,4,5,6,7,8,9,10,11,12,13,0,0,0],[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]]
```

应用场景：

在自然语言处理（NLP）中，输入的句子长度通常不同。为了让它们在一个 batch 中处理，我们需要将它们填充（pad）到相同的长度。填充的方式通常是在句子的末尾添加零（0），直到所有句子长度相同。

定义

```
struct PadParam {
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
tmp_out	[token_num, hidden_dim]	float16	ND	每一个token对应的embedding向量。
padding_offset	[1, token_num]	int32	ND	由seq_len和cum_offsets_out组合计算得到。
seq_len	[batch, 1]	int32	ND	每个batch中的有效token数量。
input_ids	[batch, max_seq_len]	int64	ND	经过pad（末尾填充0）之后，batch个token id序列。

输出

参数	维度	数据类型	格式	描述
output	[batch, hidden_dim]	float16	ND	取出的每个batch最后一个有效token的embedding向量。

 版权所有 © 华为技术有限公司

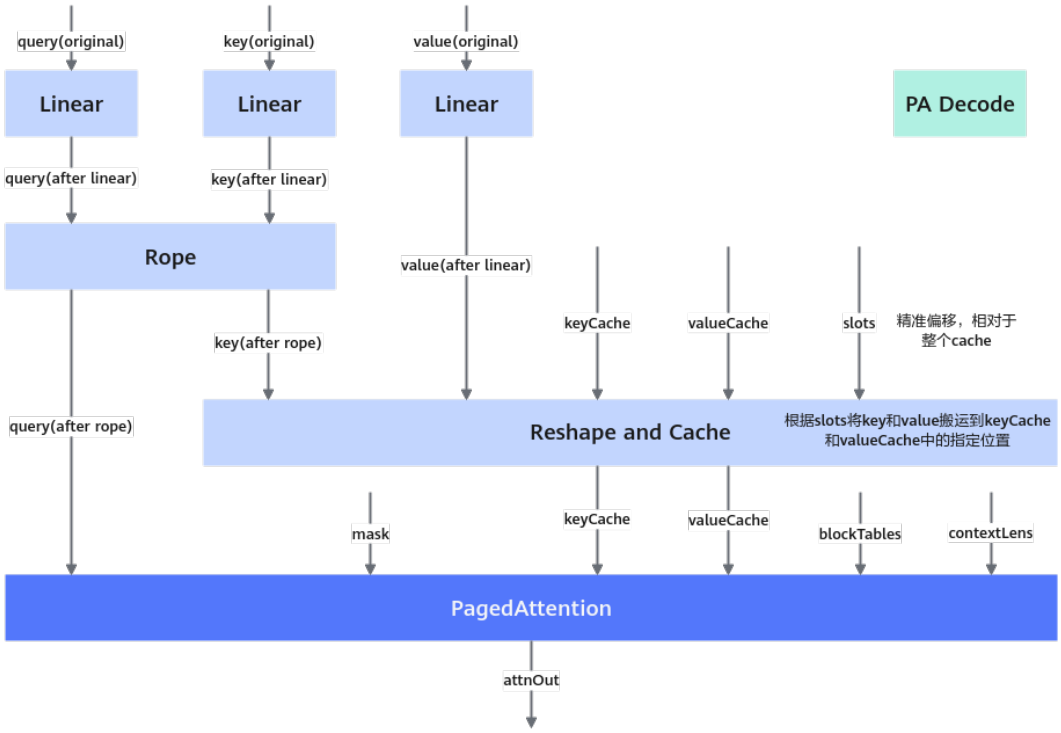
10.45 PagedAttentionOperation

10.45.1 功能描述

基于paged attention 对于kvcache分页管理的机制计算attention score。

10.45.2 算子上下文

图 10-46 paged attention 的增量阶段在模型中的位置



10.45.3 计算公式

$$PreProcess(Q, K, V, mask)$$

$$attenOut = Softmax(Zoom(Q * K^T) + mask) * V$$

*PreProcess*是对*Q*、*K*、*V*、*mask*的前置处理，如开启反量化融合功能时计算前要对输入*K*、*V*进行反量化。

*attenOut*由*attention*计算得出，其中*Zoom*为对 $Q*K^T$ 的缩放，标准计算流程中是乘以1/*sqrt*(*head_size*)，当*logN*功能开启时，额外乘以*logN*的系数。

10.45.4 定义

```
struct PagedAttentionParam {
    int32_t headNum = 0;
```

 版权所有 © 华为技术有限公司

```
    float qkScale = 1.0;
    int32_t kvHeadNum = 0;
    enum MaskType : int {
        UNDEFINED = 0,
        MASK_TYPE_NORM,
        MASK_TYPE_ALIBI,
        MASK_TYPE_SPEC
    };
    MaskType maskType = UNDEFINED;
    bool batchRunStatusEnable = false;
    enum QuantType : int {
        TYPE_QUANT_UNDEFINED = 0,
        TYPE_QUANT_UNQUANT = 0,
        TYPE_DEQUANT_FUSION,
        TYPE_QUANT_QKV_OFFLINE,
        TYPE_QUANT_QKV_ONLINE
    };
    QuantType quantType = TYPE_QUANT_UNQUANT;
    aclDataType outDataType = ACL_DT_UNDEFINED;
    bool hasQuantOffset = false;
    enum CompressType : int {
        COMPRESS_TYPE_UNDEFINED = 0,
        COMPRESS_TYPE_KVHEAD,
        COMPRESS_TYPE_KVHEAD_ROPE,
        COMPRESS_TYPE_MAX
    };
    CompressType compressType = COMPRESS_TYPE_UNDEFINED;
    enum CalcType : int {
        CALC_TYPE_UNDEFINED = 0,
        CALC_TYPE_SPEC
    };
    CalcType calcType = CALC_TYPE_UNDEFINED;
    enum ScaleType : int {
        SCALE_TYPE_TOR = 0,
        SCALE_TYPE_LOGN,
        SCALE_TYPE_MAX
    };
    ScaleType scaleType = SCALE_TYPE_TOR;
    InputLayout inputLayout = TYPE_BSND;
    uint32_t mlaVHeadSize = 0;
    uint8_t rsv[68] = {0};
};
```

10.45.5 参数列表

在使用时应带有命名空间atb::infer::PagedAttentionParam的参数，成员描述如下：

成员名称	类型	默认值	描述
headNum	int32_t	0	query 头大小。此默认值不可用，用户需配置此项参数。
qkScale	float	1.0	算子tor值。
kvHeadNum	int32_t	0	kv头数量。
maskType	MaskType	UNDEFINED	mask类型。
batchRunStatusEnable	bool	false	是否开启控制可计算batch。
quantType	QuantType	TYPE_QUANT_UNQUANT	量化类型。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
outDataType	aclDataType	ACL_DT_UNDEFINED	output数据类型。默认值表示根据输入tensors自动推导输出tensors数据类型。
hasQuantOffset	bool	false	开启量化功能后是否使用offset。
compressType	CompressType	COMPRESS_TYPE_UNDEFINED	压缩方式。
calcType	CalcType	CALC_TYPE_UNDEFINED	计算类型。
scaleType	ScaleType	SCALE_TYPE_TOR	scale类型：为SCALE_TYPE_LOGN时，不支持quanttype为2和3。
inputLayout	InputLayout	TYPE_BSND	数据排布格式默认为BSND。
mlaVHeadSize	uint32_t	0	大于0时开启MLA合并kvcache功能，表示kv合并传入时v的head_size取值范围为[0,576]。
rsv[68]	uint8_t	{0}	预留参数。

上表中类型为自定义类型的，其描述如下：

- **MaskType**：表示mask类型的枚举量，其具体取值如下。maskType取不同MaskType值或在不同硬件上运行时维度不同，详见**10.45.8.2 mask类型**。
  - UNDEFINED（枚举值：0）：默认值，全0的mask。
  - MASK_TYPE_NORM（枚举值：1）：倒三角mask。
  - MASK_TYPE_ALIBI（枚举值：2）：alibi mask。
  - MASK_TYPE_SPEC（枚举值：3）：并行解码mask。
- **QuantType**：表示量化类型的枚举量，其具体取值如下：
  - TYPE_QUANT_UNDEFINED（枚举值：0）：默认值，不与量化融合。
  - TYPE_QUANT_UNQUANT（枚举值：0）：默认值，不与量化融合。
  - TYPE_DEQUANT_FUSION（枚举值：1）：与反量化融合，只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
  - TYPE_QUANT_QKV_OFFLINE（枚举值：2）：离线INT8量化，只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
  - TYPE_QUANT_QKV_ONLINE（枚举值：3）：在线INT8量化，只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
  - 量化类型有以下约束条件：
    - 为TYPE_QUANT_UNQUANT时，q、keyCache、valueCache为bf16/float16。

 版权所有 © 华为技术有限公司

        - 为TYPE_DEQUANT_FUSION时，q为bf16/float16，keyCache、valueCache为int8。
        - 为TYPE_QUANT_QKV_OFFLINE或TYPE_QUANT_QKV_ONLINE时，q、keyCache、valueCache为int8。
        - keyCache，valueCache的headsize等长，范围为（0, 256]，且block_size * head_size ≤ 128 * 128。
        - 为TYPE_QUANT_QKV_OFFLINE或TYPE_QUANT_QKV_ONLINE时，outdatatype需要配置，只能是ACL_FLOAT16或ACL_BF16。inputLayout只支持TYPE_BSND。
- **CompressType**：表示压缩类型的枚举量，其具体取值如下：
  - COMPRESS_TYPE_UNDEFINED（枚举值：0）：默认值，不压缩。
  - COMPRESS_TYPE_KVHEAD（枚举值：1）：压缩key_cache, value_cache的kvHead维度，只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
  - COMPRESS_TYPE_KVHEAD_ROPE（枚举值：2）：rope场景压缩key_cache, value_cache的kvHead维度，只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
  - COMPRESS_TYPE_MAX（枚举值：3）：压缩类型边界值，仅用于判断是否出界，所有情况不能取该值。
  - 压缩类型有以下约束条件：
    - 压缩方式为COMPRESS_TYPE_KVHEAD时，不支持quanttype为2和3。
    - 压缩方式为COMPRESS_TYPE_KVHEAD_ROPE时, maskType需传0。此时不支持quanttype为2和3。
- **CalcType**：表示计算类型的枚举量，其具体取值如下：
  - CALC_TYPE_UNDEFINED（枚举值：0）：默认值，不开启并行解码。
  - CALC_TYPE_SPEC（枚举值：1）：并行解码功能，此时只支持quantType = 0。
- **ScaleType**：表示是否开启LogN特性的枚举量，其具体取值如下：
  - SCALE_TYPE_TOR（枚举值：0）：默认值，不开启LogN缩放。
  - SCALE_TYPE_LOGN（枚举值：1）：注意力使用LogN缩放。
  - SCALE_TYPE_MAX（枚举值：2）：边界值，仅用于判断是否出界。
  - scale类型有以下约束条件：

    “scale”类型为SCALE_TYPE_LOGN时，不支持quanttype为2和3。
- **InputLayout**：表示数据排布格式的枚举量，其具体取值如下：
  - TYPE_BSND（枚举值：0）：默认值，表示数据排布为BSND。
  - TYPE_BNSD（枚举值：1）：表示数据排布为BNSD。

10.45.6 输入输出列表

列表中的shape仅为通用场景下的shape。特殊场景的shape见**10.45.8 功能列表**。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
query	[num_tokens, num_head, head_size]	float16/ bf16/ int8	ND	npu	各batch的query在num_tokens轴合并；当开启全量化功能时仅支持int8。	基础场景
keyCache	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：[num_blocks, block_size, kv_head_num, head_size_k]; • Atlas 推理系列产品：[num_blocks, head_size *num_heads / 16 ,block_size, 16]	float16/ bf16/ int8	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：ND • Atlas 推理系列产品：NZ	npu	cache好的key；当开启反量化融合功能和全量化功能时，只支持int8。	基础场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
valueCache	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：[num_blocks, block_size, kv_head_num, head_size_v]; • Atlas 推理系列产品：[num_blocks, head_size *num_heads / 16 ,block_size, 16]	float16/bf16/int8	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：ND • Atlas 推理系列产品：NZ	npu	cache好的value；当开启反量化融合功能和全量化功能时，只支持int8。当开启mla合并输出kvcache功能时不传入。	基础场景
blockTables	[num_tokens, max_num_blocks_per_query]	int32	ND	npu	每个query的kvcache的block table，第一维是token索引，第二维表示block索引。	基础场景
contextLens	[batch]	int32	ND	cpu	每个query对应的key/value的token数量。	基础场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
mask	较复杂，见功能列表 **10.45.8.2 mask类型**	float16/bf16	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：ND • Atlas 推理系列产品：NZ	npu	Atlas 800I A2 推理产品/Atlas A2 训练系列产品：当“maskType”不为UNDEFINED时输入。	mask场景
batchRunStatus	[batch]	int32	ND	cpu	当开启控制可计算batch功能时需要传此tensor以作为哪些batch参与计算的标志位。	控制可计算batch场景
kDescale	[k_head_num*head_size]	int64/float	ND	npu	当量化类型为反量化时，步长tensor；数据类型需与vDescale保持一致。	反量化融合场景或全量化场景
kOffset	[k_head_num*head_size]	int32	ND	npu	当量化类型为反量化，且“hasQuantOffset”为true时，输入k的偏移量。	非对称反量化场景
vDescale	[v_head_num*head_size]	int64/float	ND	npu	当量化类型为反量化时，步长tensor。	反量化融合场景或全量化场景
vOffset	[v_head_num*head_size]	int32	ND	npu	当量化类型为反量化，且“hasQuantOffset”为true时，输入v的偏移量。	非对称反量化场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
qSeqLens	[batch]	int32	ND	cpu	当开启并行解码功能时需要传此tensor，每个batch对应的seqLen。	并行解码场景
razorOffset	[num_blocks, block_size]	float	ND	npu	当开启Razor Rope功能时需要传此tensor。	多头自适应压缩（rope场景）
pScale	[q_head_num]	float	ND	npu	当开启离线全量化时需要传此tensor，传入attention计算中间值p_score的量化scale。	离线全量化场景
logN	[batch]	float	ND	npu	当scale类型为logN时需要传此tensor，为各batch对应的缩放系数。	LogN场景
attnOut	[num_tokens, num_head, head_size_v]	float16/bf16	ND	npu	经过计算输出的query。	基础场景

上表中，contextLens和qSeqLens较为特殊，其虽为输入tensor，在使用时却是当做param使用，故需要有一个bind tensor的操作，使其从device侧提取出来绑定到host侧。contextLens在Atlas 800I A2 推理产品/Atlas A2 训练系列产品的runner中为CPU侧，在非Atlas 800I A2 推理产品/Atlas A2 训练系列产品的runner中为NPU侧。

上表中，使用场景表示各个输入tensor所对应的场景，其中基础场景是指该tensor为必须项，PA所有情况均有此tensor，其中包含5个输入tensor和1个输出tensor。而属于其他使用场景的输入tensor则是在对应场景下才能够传此输入tensor，非对应场景下不能传此输入tensor。以下为不同场景的定义：

- mask场景指“maskType”不为UNDEFINED时。
- 控制可计算batch场景指“batchRunStatusEnable”为True时。
- 反量化融合场景指“quantType”为TYPE_DEQUANT_FUSION时。
- 全量化场景指“quantType”为TYPE_QUANT_QKV_OFFLINE（离线全量化场景）或TYPE_QUANT_QKV_ONLINE（在线全量化场景）时。

 版权所有 © 华为技术有限公司

- 非对称反量化场景指“quantType”为TYPE_DEQUANT_FUSION且“hasQuantOffset”为True时。
- 并行解码场景指“calcType”为CALC_TYPE_SPEC时。
- 多头自适应压缩（rope场景）指“compressType”为COMPRESS_TYPE_KVHEAD_ROPE时。
- 离线全量化场景指“quantType”为TYPE_QUANT_QKV_OFFLINE时。
- LogN场景指“scaleType”为SCALE_TYPE_LOGN时。

10.45.7 规格约束

- 因硬件限制，block_size %16 == 0，推荐 block_size = 128。
- 开启并行解码功能时，blockSize <= 128。
- blockTables中元素的值须在[0, num_blocks)之间。
- head_size范围为（0, 256]，keyCache和valueCache的格式为NZ时，head_size必须是16倍数。
- 在Atlas 推理系列产品上：
  - query、keyCache、valueCache、maskTensor 四个入参只支持float16。
  - 0<batch<=2048。
- 多头自适应压缩、并行解码场景、量化场景、以及Atlas 推理系列产品上，keyCache，valueCache的headsize等长，范围为（0, 256]，且block_size * head_size ≤ 128 * 128，否则keyCache，valueCache的head_size可以不相同，范围为（0, 576]，当keyCache或valueCache的head_size > 256时，block_size小于等于128。
- 针对mask类型，如果原模型非alibi，且mask没有做padding，可选择“maskType”为UNDEFINED，即不传mask；其他情况则依据原生模型的配置，选择“maskType”是MASK_TYPE_NORM或MASK_TYPE_ALIBI。
- BNSD不支持PA_ENCODER。
- Atlas 训练系列产品与Atlas 推理系列产品的输入输出约束相同，Atlas 训练系列产品只支持“maskType”为UNDEFINED、MASK_TYPE_NORM或MASK_TYPE_ALIBI的基础功能场景，其余特性参数需为默认值。
- 若想使用GQA模式，需满足headNum > kvHeadNum且headNum%kvHeadNum == 0。
- 当blocksize = head_size = 256时，在Atlas 800I A2 推理产品/Atlas A2 训练系列产品上，输入格式需满足BSND、非量化、decoder场景。

10.45.8 功能列表

10.45.8.1 各功能共存情况

Y表示两功能不冲突，N表示两功能冲突。

 版权所有 © 华为技术有限公司

	控制可计算batch	多头自适应压缩（alibi场景）	多头自适应压缩（rope场景）	并行解码	MLA	反量化融合	qkv全量化	logN缩放	BNSD维度输入	MLA合并输入kvcache功能	MLA并行解码
控制可计算batch	-	Y	Y	N	Y	Y	Y	Y	Y	Y	N
多头自适应压缩（alibi场景）	-	-	N	N	N	Y	N	N	N	N	N
多头自适应压缩（rope场景）	-	-	-	N	N	Y	N	N	N	N	N
并行解码	-	-	-	-	N	N	N	N	N	N	-
MLA	-	-	-	-	-	N	N	N	N	-	-
反量化融合	-	-	-	-	-	-	N	N	N	N	N
qkv全量化	-	-	-	-	-	-	-	N	N	Y	N
logN缩放	-	-	-	-	-	-	-	-		N	N
BNSD维度输入	-	-	-	-	-	-	-	-	-	N	N
MLA合并输入kvcache功能	-	-	-	-	-	-	-	-	-	-	-
MLA并行解码	-	-	-	-	-	-	-	-	-	-	-

10.45.8.2 mask 类型

说明

mask类型不是独立特性，因paged attention算子的mask较为复杂，为便于理解，此处单独针对mask类型进行说明。

maskType	硬件类型	维度	备注
UNDEFINED	不传mask	不传mask	相当于一个全零的mask。
MASK_TYPE_NORM	Atlas 800I A2 推理产品/Atlas A2 训练系列产品	[batch, 1, max_seq_len] 或 [1, max_seq_len] 或 [max_seq_len, max_seq_len]	倒三角mask。
	Atlas 推理系列产品	[batch, max_seq_len / 16, 16, 16] 或 [1, max_seq_len / 16, 16, 16]	
MASK_TYPE_ALIBI	Atlas 800I A2 推理产品/Atlas A2 训练系列产品	[batch, num_head, 1, max_seq_len] 或 [num_head, 1, max_seq_len]	alibi mask。
	Atlas 推理系列产品	[batch * num_head, max_seq_len / 16, 16, 16] 或 [num_head, max_seq_len / 16, 16, 16]	
MASK_TYPE_SPEC	Atlas 800I A2 推理产品/Atlas A2 训练系列产品	[num_tokens, max_seq_len]	并行解码mask。

 版权所有 © 华为技术有限公司

maskType	硬件类型	维度	备注
	Atlas 推理系列产品	[1, max_seq_len / 16, num_tokens, 16]	

上表中Atlas 推理系列产品上max_seq_len应16对齐，且维度描述中的除法均为ceil div。

10.45.8.3 基础功能

当参数maskType、batchRunStatusEnable、quantType、calcType、compressType、scaleType均为默认值时，使用PagedAttention的基础功能，此时输入输出参数如下：

参数	维度	数据类型	格式	cpu or npu	描述
query	[num_tokens, num_head, head_size]	float16/ bf16	ND	npu	各batch的query在num_tokens轴合并。
keyCache	• Atlas 800I A2 推理产品/ Atlas A2 训练系列产品：[num_blocks, block_size, kv_head_num , head_size_k]; • Atlas 推理系列产品：[num_blocks, head_size *num_heads / 16 ,block_size , 16]	float16/ bf16	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：ND • Atlas 推理系列产品：NZ	npu	cache好的key。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu or npu	描述
valueCache	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：[num_blocks, block_size, kv_head_num, head_size_v]; • Atlas 推理系列产品：[num_blocks, head_size *num_heads / 16 ,block_size , 16]	float16/bf16	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：ND • Atlas 推理系列产品：NZ	npu	cache好的value。
blockTables	[num_tokens, max_num_blocks_per_query]	int32	ND	npu	每个query的kvcache的block table，第一维是token索引，第二维表示block索引。
contextLens	[batch]	int32	ND	cpu	每个query对应的key/value的token数量。
attnOut	[num_tokens, num_head, head_size_v]	float16/bf16	ND	npu	经过计算输出的query。

10.45.8.4 控制可计算 batch

功能

指定某几个batch参与attention计算。

开启方式

参数“batchRunStatusEnable”置为true，并传入batchRunStatus作为输入tensor。

batchRunStatus为0，1组成的tensor。0代表该位置的batch不参与计算，1代表参与计算。

 版权所有 © 华为技术有限公司

特殊约束

不支持Atlas 推理系列产品。

10.45.8.5 多头自适应压缩（alibi 场景）

功能

alibi mask场景下，对kv的 head_num维度进行压缩，提高内存利用率，精度不变。

又名Razor Attention。

开启方式

参数“compressType”置为COMPRESS_TYPE_KVHEAD。

若干输入tensor需要传入指定的维度，如下：

参数	维度	数据类型	格式	cpu or npu
query	[num_tokens * kv_head_num, num_head / kv_head_num, head_size]	float16/ bf16	ND	npu
keyCache	[num_blocks, block_size, 1, head_size]	float16/ bf16/int8	ND	npu
valueCache	[num_blocks, block_size, 1, head_size]	float16/ bf16/int8	ND	npu
blockTables	[num_tokens * kv_head_num, max_num_blocks_per_query]	int32	ND	npu
contextLens	[batch * num_head]	int32	ND	cpu
attnOut	[n_tokens, num_head, head_size]	float16/ bf16	ND	npu

特殊约束

不支持Atlas 推理系列产品。

需与ReshapeAndCache算子的相应功能配合使用。

10.45.8.6 多头自适应压缩（rope 场景）

功能

rope mask场景下，对kv的 head_num维度进行压缩，提高内存利用率，精度不变。

又名Razor Attention。

 版权所有 © 华为技术有限公司

开启方式

参数“compressType”置为COMPRESS_TYPE_KVHEAD_ROPE。

额外传入razorOffset作为输入tensor。

若干输入tensor需要传入指定的维度，如下：

参数	维度	数据类型	格式	cpu or npu
query	[num_tokens * kv_head_num, num_head / kv_head_num, head_size]	float16/ bf16	ND	npu
keyCache	[num_blocks, block_size, 1, head_size]	float16/ bf16/int8	ND	npu
valueCache	[num_blocks, block_size, 1, head_size]	float16/ bf16/int8	ND	npu
blockTables	[num_tokens * kv_head_num, max_num_blocks_per_query]	int32	ND	npu
contextLens	[batch * num_head]	int32	ND	cpu
razorOffset	[num_tokens * kv_head_num, block_size]	float	ND	cpu
attnOut	[n_tokens, num_head, head_size]	float16/ bf16	ND	npu

特殊约束

不支持Atlas 推理系列产品。

需与ReshapeAndCache算子的相应功能配合使用。

10.45.8.7 并行解码

功能

在传统的推理过程中，需要逐个token的进行串行解码，导致时间消耗与生成的token数量成正比，这个缺点在实施逐步解码的情况下尤为明显，为了增强这个过程，引入了lookahead。在decode阶段从n-gram中获取多个候选Token，进行并行解码，提升模型的推理速度。

开启方式

额外传入qSeqLens作为输入tensor，变量“calcType”置为CALC_TYPE_SPEC。

- Atlas 800I A2 推理产品/Atlas A2 训练系列产品：变量“maskType”置为MASK_TYPE_NORM或MASK_TYPE_SPEC。

 版权所有 © 华为技术有限公司

- Atlas 推理系列产品：变量“maskType”置为UNDEFINED或MASK_TYPE_SPEC。

特殊约束

无

10.45.8.8 MLA

功能

DeepSeek V2提出Multi-Head Latent Attention，通过对hiddenstates进行down-projection处理，压缩hiddensize，减少KVCache显存的占用。通过up-projection恢复KV，进行PA计算。通过算力换显存，来提升吞吐。

开启方式

keyCache的维度[num_blocks, block_size, kv_head_num, head_size_k]与valueCache的维度[num_blocks, block_size, kv_head_num, head_size_v]中head_size_k与head_size_v不相等。

特殊约束

不支持Atlas 推理系列产品。

10.45.8.9 反量化融合

功能

支持量化好的k，v传入，降低显存占用。由于传入的q不是量化的，故又名伪量化。

开启方式

参数“quantType”置为TYPE_DEQUANT_FUSION。

当以上参数开启时，通过参数hasQuantOffset指明传入的量化后的kv是否需要相应的偏移量。

特殊约束

不支持Atlas 推理系列产品。

10.45.8.10 qkv 全量化

功能

支持量化好的q、k、v传入，降低显存占用。

开启方式

参数“quantType”置为TYPE_QUANT_QKV_OFFLINE或TYPE_QUANT_QKV_ONLINE, 分别为离线量化与在线量化。

 版权所有 © 华为技术有限公司

复用kDescale和vDescale，当采用离线量化时，额外传入pScale作为输入tensor。

若干输入tensor需要传入指定的维度，如下：

参数	维度	数据类型	格式	cpu or npu
query	[num_tokens, num_head, head_size]	int8	ND	npu
keyCache	[num_blocks, block_size, kv_head_num, head_size]	int8	ND	npu
valueCache	[num_blocks, block_size, kv_head_num, head_size]	int8	ND	npu
blockTables	[num_tokens, max_num_blocks_per_query]	int32	ND	npu
contextLens	[batch]	int32	ND	cpu
kDescale	[head_num]	float	ND	npu
vDescale	[head_num]	float	ND	npu
pScale	[head_num]	float	ND	npu
attnOut	[n_tokens, num_head, head_size]	float16/ bf16	ND	npu

其中pScale仅在离线量化场景传入，在线量化场景则不传此输入tensor。

特殊约束

使用全量化时需要指定输出tensor的数据类型，具体为使用参数outDataType，该参数只能是ACL_FLOAT16或ACL_BF16。

不支持Atlas 推理系列产品。

10.45.8.11 logN 缩放

功能

针对Qwen长序列场景，使用LogN缩放注意力。

其公式为：

$$Attention\left(Q, K, V \right) = softmax\left(\frac{log_m n}{\sqrt{d}} QK^T \right) V$$

 版权所有 © 华为技术有限公司

其中：m为训练长度，n为预测长度。

开启方式

将参数“scaleType置”为infer::PagedAttentionParam::SCALE_TYPE_LOGN时即可使用LogN缩放。

使用LogN缩放时，PagedAttentionOperation 新增一个 输入tensor 名为 logN，其具体规格如下：

名字	维度	数据类型	格式	cpu or npu
logN	增量阶段: [batch size]	Atlas 800I A2 推理产品/Atlas A2 训练系列产品：float32； Atlas 推理系列产品：float16	ND	NPU

其它输入tensor没有变化。

特殊约束

无

10.45.8.12 BNSD 维度输入

功能

针对X1场景，使PA Op支持BNSD数据排布（原有的数据排布格式为BSND）。

开启方式

将参数“inputLayout”置为infer::inputLayout::TYPE_BNSD时开启BNSD功能。该参数默认值为infer::inputLayout::TYPE_BSND，表示默认数据排布格式为BSND。

若干tensor的规格发生了改变，改变的tensor如下表所示：

参数	维度	数据类型	格式	cpu or npu
q	[num_tokens, num_head, head_dim]	float16/ bf16	ND	npu
kCache	[num_blocks, kv_head_num, block_size, head_size]	float16/ bf16	ND	npu
vCache	[num_blocks, kv_head_num, block_size, head_size]	float16/ bf16	ND	npu
out	[num_tokens, num_head, head_dim]	float16/ bf16	ND	npu

 版权所有 © 华为技术有限公司

num_tokens = batch * qseqlen，当前qseqlen = 1。

其余tensor未发生改变。

特殊约束

无

10.45.8.13 MLA 合并输入 kvcache 功能

功能

过去attention算子会将k和v存储在显存中，随着模型越来越大，输入序列越来越长，kvcache也越来越大，对显存的容量造成很大压力，造成性能上的瓶颈。因此Multi-Head Latent Attention提出将kvcache压缩成一个一个较小的向量，代替原来的kvcache输入进fa算子进行计算。

开启方式

mlaVHeadSize > 0

特殊约束

- MLA合并输入kvcache功能不支持Atlas 推理系列产品，不支持alibi mask，多头自适应压缩（rope 场景），多头自适应压缩（alibi 场景），并行解码，logN缩放，BNSD输入排布。
- 开启MLA合并输入kvcache功能后query和keyCache的head_size范围为（0, 576]，当keyCache或valueCache的head_size > 256时，block_size小于等于128，mlaVHeadSize不能大于query和keyCache的head_size。
- MLA合并kvcache功能支持全量化场景，一起开启时query和key的head_size范围为（0, 576]，mlaVHeadSize不能大于query和key的head_size，当keyCache或valueCache的head_size > 256时，block_size小于等于128。
- MLA合并输入只支持MQA场景。
- Atlas 800I A2 推理产品/Atlas A2 训练系列产品上0<batch<=8192。

10.45.8.14 MLA 并行解码

功能

在MLA场景下的并行解码功能。

开启方式

- “calcType”置为CALC_TYPE_SPEC
- mlaVHeadSize > 0

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	cpu or npu	描述	使用场景
query	[num_tokens, num_head, head_size]	float16/bf16	ND	npu	各batch的query在num_tokens轴合并。	基础场景
keyCache	[num_blocks, block_size, kv_head_num, head_size_k]	float16/bf16	ND	npu	kcache。	基础场景
blockTables	[num_tokens, max_num_blocks_per_query]	int32	ND	npu	每个query的kvcache的block table，第一维是token索引，第二维表示block索引。	基础场景
contextLens	[batch]	int32	ND	cpu	每个query对应的key/value的token数量。	基础场景
mask	[num_tokens(合轴)，max_seqlen]	float16/bf16	ND	npu	当maskType不为UNDEFINED时输入。	mask场景
qSeqLens	[batch]	int32	ND	cpu	当开启并行解码功能时需要传此tensor，每个batch对应的seqLen。	并行解码场景

输出

参数	维度	数据类型	格式	cpu or npu	描述	使用场景
attnOut	[num_tokens, num_head, head_size_v]	float16/bf16	ND	npu	经过计算输出的query。	基础场景

 版权所有 © 华为技术有限公司

特殊约束

- 与MLA合并输入的约束相同（embed大小限制）。
- qSeqlen中的值大于等于1，MTP需要不全为1。
- maskType支持UNDEFINED或者MASK_TYPE_SPEC。
- MLA下仅支持MQA场景。
- head_num范围为(0,256]，head_size、head_size_k范围为(0,576]，blocksize范围为(0,128]。

10.45.9 算子调用示例（C++）

10.45.9.1 paged_attention_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：mask场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include <stdlib.h>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const uint32_t NTOKENS = 2;                         // token数量
const uint32_t BATCH_SIZE = NTOKENS;                    // batch数量
const uint32_t MAX_SEQ_LEN = 1024;                  // 最大序列长度
const uint32_t HEAD_NUM = 32;                       // 头数
const uint32_t KV_HEAD_NUM = 32;                    // kv头数
const uint32_t HEAD_SIZE = 128;                     // 头大小
const uint32_t BLOCK_NUM = 16;                      // 块数量
const uint32_t BLOCK_SIZE = 128;                    // 块大小
const uint32_t MAX_CONTEXT_LEN = 1024;              // 上下文最大长度
std::vector<int32_t> contextLensData(BATCH_SIZE, 256);  // contextLens的host侧数据

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建query tensor
    std::vector<float> queryData(NTOKENS * HEAD_NUM * HEAD_SIZE, 1.0);
    atb::Tensor query = CreateTensorFromVector(
        contextPtr, stream, queryData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, HEAD_NUM,
HEAD_SIZE});
    // 创建key，value tensor
    std::vector<float> kvCacheData(BLOCK_NUM * BLOCK_SIZE * KV_HEAD_NUM * HEAD_SIZE, 1.0);
    atb::Tensor kCache = CreateTensorFromVector(contextPtr,
        stream,
        kvCacheData,
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {BLOCK_NUM, BLOCK_SIZE, KV_HEAD_NUM, HEAD_SIZE});
    atb::Tensor vCache = CreateTensorFromVector(contextPtr,
```

```
        stream,
        kvCacheData,
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {BLOCK_NUM, BLOCK_SIZE, KV_HEAD_NUM, HEAD_SIZE});
    // 创建blockTables
    uint32_t maxNumBlocksPerQuery = (MAX_CONTEXT_LEN + BLOCK_SIZE - 1) / BLOCK_SIZE;
    std::vector<int32_t> blockTablesData(NTOKENS * maxNumBlocksPerQuery, 0);
    for (size_t i = 0; i < blockTablesData.size(); i++) {
        blockTablesData.at(i) = rand() % (BLOCK_NUM - 1);
    }
    atb::Tensor blockTables = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {NTOKENS,
maxNumBlocksPerQuery});
    CHECK_STATUS(aclrtMemcpy(blockTables.deviceData,
        blockTables.dataSize,
        blockTablesData.data(),
        sizeof(int32_t) * blockTablesData.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    // 创建contextLens，host侧tensor
    atb::Tensor contextLens = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    contextLens.hostData = contextLensData.data();
    // 创建norm mask，值为-inf的上三角mask
    std::vector<float> maskData(BATCH_SIZE * MAX_SEQ_LEN, 0);
    for (int i = 0; i < BATCH_SIZE; ++i) {
        for (int j = 0; j < MAX_SEQ_LEN; ++j) {
            maskData[i * MAX_SEQ_LEN + j] = -32768;  // 32768 : -inf
        }
    }
    atb::Tensor mask = CreateTensorFromVector(
        contextPtr, stream, maskData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE, 1,
MAX_SEQ_LEN});
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {query, kCache, vCache, blockTables, contextLens, mask};
    return inTensors;
}

/**
 * @brief 创建一个PA的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::PagedAttentionParam paOpParam;
    paOpParam.maskType = atb::infer::PagedAttentionParam::MaskType::MASK_TYPE_NORM;
    paOpParam.headNum = HEAD_NUM;
    paOpParam.kvHeadNum = KV_HEAD_NUM;
    paOpParam.qkScale = 0.08838834764831843;
    atb::Operation *paOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(paOpParam, &paOp));
    return paOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // PA示例
    atb::Operation *paOp = PrepareOperation();
    // 准备输入张量
    atb::VariantPack paVariantPack;
    paVariantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
```

 版权所有 © 华为技术有限公司

```
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS,
HEAD_NUM, HEAD_SIZE});
    paVariantPack.outTensors.push_back(tensorOut);  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(paOp->Setup(paVariantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // PA执行
    paOp->Execute(paVariantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    CHECK_STATUS(aclrtFree(tensorOut.deviceData));
    for (atb::Tensor &inTensor : paVariantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(paOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "PA demo success!" << std::endl;
    return 0;
}
```

10.45.9.2 paged_attetion_inference_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。本用例仅支持Atlas 推理系列产品。

与示例1相较本示例主要有以下修改点：

- kvCache shape改为：[block_num, head_size * head_num / 16, block_size, 16]。
- kvCache数据格式改为：ACL_FORMAT_FRACTAL_NZ。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include <stdlib.h>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const uint32_t NTOKENS = 2;                         // token数量
const uint32_t BATCH_SIZE = NTOKENS;                    // batch数量
const uint32_t MAX_SEQ_LEN = 1024;                  // 最大序列长度
const uint32_t HEAD_NUM = 32;                       // 头数
const uint32_t KV_HEAD_NUM = 32;                    // kv头数
const uint32_t HEAD_SIZE = 128;                     // 头大小
const uint32_t BLOCK_NUM = 16;                      // 块数量
const uint32_t BLOCK_SIZE = 128;                    // 块大小
const uint32_t MAX_CONTEXT_LEN = 1024;              // 上下文最大长度
std::vector<int32_t> contextLensData(BATCH_SIZE, 256);  // contextLens的host侧数据

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
```

 版权所有 © 华为技术有限公司

```
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建query tensor
    std::vector<unsigned int16_t> queryData(NTOKENS * HEAD_NUM * HEAD_SIZE, 0x3C00); // 0x3C00:
float16的1
    atb::Tensor query = CreateTensorFromVector(
        contextPtr, stream, queryData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, HEAD_NUM,
HEAD_SIZE}, ACL_FLOAT16);
    // 创建key，value tensor
    std::vector<unsigned int16_t> kvCacheData(BLOCK_NUM * BLOCK_SIZE * KV_HEAD_NUM * HEAD_SIZE,
0x3C00);
    atb::Tensor kCache = CreateTensorFromVector(contextPtr,
        stream,
        kvCacheData,
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_FRACTAL_NZ,
        {BLOCK_NUM, HEAD_SIZE *KV_HEAD_NUM / 16, BLOCK_SIZE, 16}, ACL_FLOAT16);
    atb::Tensor vCache = CreateTensorFromVector(contextPtr,
        stream,
        kvCacheData,
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_FRACTAL_NZ,
        {BLOCK_NUM, HEAD_SIZE *KV_HEAD_NUM / 16, BLOCK_SIZE, 16}, ACL_FLOAT16);
    // 创建blockTables
    uint32_t maxNumBlocksPerQuery = (MAX_CONTEXT_LEN + BLOCK_SIZE - 1) / BLOCK_SIZE;
    std::vector<int32_t> blockTablesData(NTOKENS * maxNumBlocksPerQuery, 0);
    for (size_t i = 0; i < blockTablesData.size(); i++) {
        blockTablesData.at(i) = rand() % (BLOCK_NUM - 1);
    }
    atb::Tensor blockTables = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {NTOKENS,
maxNumBlocksPerQuery});
    CHECK_STATUS(aclrtMemcpy(blockTables.deviceData,
        blockTables.dataSize,
        blockTablesData.data(),
        sizeof(int32_t) * blockTablesData.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    // 创建contextLens，host侧tensor
    atb::Tensor contextLens = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    contextLens.hostData = contextLensData.data();
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {query, kCache, vCache, blockTables, contextLens};
    return inTensors;
}

/**
 * @brief 创建一个PA的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::PagedAttentionParam paOpParam;
    paOpParam.headNum = HEAD_NUM;
    paOpParam.kvHeadNum = KV_HEAD_NUM;
    paOpParam.qkScale = 0.08838834764831843;
    atb::Operation *paOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(paOpParam, &paOp));
    return paOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
```

 版权所有 © 华为技术有限公司

```
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // PA示例
    atb::Operation *paOp = PrepareOperation();
    // 准备输入张量
    atb::VariantPack paVariantPack;
    paVariantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS,
HEAD_NUM, HEAD_SIZE});
    paVariantPack.outTensors.push_back(tensorOut);  // 放入输出tensor

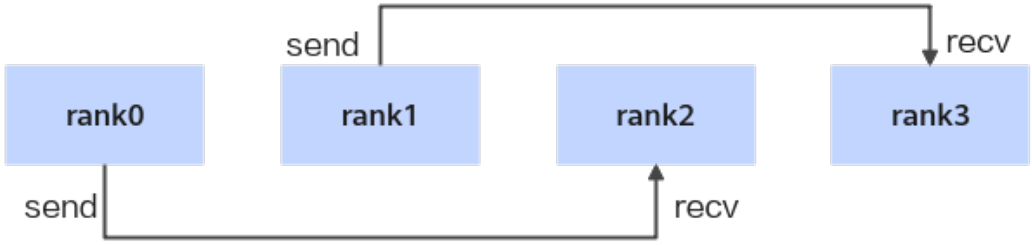
    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(paOp->Setup(paVariantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // PA执行
    paOp->Execute(paVariantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    CHECK_STATUS(aclrtFree(tensorOut.deviceData));
    for (atb::Tensor &inTensor : paVariantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(paOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "PA demo success!" << std::endl;
    return 0;
}
```

10.46 RecvOperation

功能

从当前通信卡接收来自指定通信卡的数据。Send和Recv需要配套使用。

图 10-47 Send&Recv 收发数据功能示意图



定义

```
struct RecvParam {
    int rank = 0;
```

 版权所有 © 华为技术有限公司

```
    int rankSize = 0;
    int rankRoot = 0;
    uint32_t srcRank = 1;
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rank	int	0	当前卡所属通信编号。
rankSize	int	0	通信的卡的数量。
rankRoot	int	0	主通信编号。
srcRank	uint32_t	1	通信域内数据接收端的rank编号。
backend	std::string	"hccl"	通信后端指示，仅支持"hccl"。
hcclComm	HcclComm	nullptr	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	通信模式，**CommMode**类型枚举值。hccl多线程只支持外部传入通信域方式。
rankTableFile	std::string	-	集群信息的配置文件路径，适用单机以及多机通信场景，当前仅支持hccl后端场景,若单机配置了ranktable，则以ranktable来初始化通信域。 ranktable配置请参见《CANN TensorFlow 1.15 模型迁移指南》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”。
commDomain	std::string	-	通信device组用通信域名标识，多通信域时使用，当前仅支持hccl。
rsv[64]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[d0,…,dn]	"hccl": float16/float/int8/int16/int32/int64/bf16	ND	输入tensor。

 版权所有 © 华为技术有限公司

输出

参数	维度	数据类型	格式	描述
output	[d0,…,dn]	"hccl": float16/float/int8/ int16/int32/int64/bf16	ND	输出tensor，与输入维度相同。

规格约束

- 仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- rank、rankSize、rankRoot需满足以下条件：
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize
  - 0 ≤ scrRank < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：

```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```

10.47 ReduceOperation

功能

在指定维度上求和、取最大值或最小值，并消除这个维度。

计算公式

- **REDUCE_MAX:**

  相当于pytorch中

  ```
  intensor.amax(axis)
  ```

  - **示例1**

    x shape: [2,3], axis: [0]

    ```
    x:
    tensor([[ 1, 18, 17],
            [16, 10,  1]])
    output:
    tensor([16, 18, 17])
    ```

  - **示例2**

    x shape: [2,3,2], axis: [0,1]

    ```
    x:
    tensor([[[17,  3],
             [19,  6],
             [ 1, 18]],

            [[ 4,  5],
    ```

```
        [ 8,  1],
        [14, 18]]])
output:
tensor([19, 18])
```

- **REDUCE_MIN：**

  相当于pytorch中

  ```
  intensor.amin(axis)
  ```

  - **示例1**

    x shape: [2,3], axis: [0]

    ```
    x:
    tensor([[16,  2,  7],
            [18, 16, 17]])
    output:
    tensor([16,  2,  7])
    ```

  - **示例2**

    x shape: [2,3,2], axis: [0,1]

    ```
    x:
    tensor([[[ 9,  4],
             [11,  0],
             [ 2,  6]],

            [[ 2,  7],
             [ 8, 16],
             [ 5,  3]]])
    output:
    tensor([2, 0])
    ```

- **REDUCE_SUM：**

  相当于pytorch中

  ```
  torch.sum(intensor, axis)
  ```

  - **示例1**

    x shape: [2,3], axis: [0]

    ```
    x:
    tensor([[-0.9701, -0.8059,  0.2919],
            [-0.1444, -2.3023,  1.2257]])
    output:
    tensor([-1.1145, -3.1082,  1.5176])
    ```

  - **示例2**

    x shape: [2,3,2], axis: [0,1]

    ```
    x:
    tensor([[[ 1.4630, -0.3619],
             [-0.4526, -1.0142],
             [-0.2883,  0.9334]],

            [[ 0.7801, -0.1580],
             [ 0.4843,  1.8145],
             [ 0.9706, -1.8114]]])
    output:
    tensor([ 2.9571, -0.5976])
    ```

定义

```
struct ReduceParam {
    enum ReduceType {
        REDUCE_UNDEFINED = 0,
        REDUCE_MAX,
        REDUCE_MIN,
        REDUCE_SUM,
    };
```

 版权所有 © 华为技术有限公司

```
    ReduceType reduceType = REDUCE_UNDEFINED;
    SVector<int64_t> axis;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
reduceType	ReduceType	REDUCE_UNDEFINED	计算类型，支持以下参数： ● REDUCE_UNDEFINED：未定义。 ● REDUCE_MAX：求最大值。 ● REDUCE_MIN：求最小值。 ● REDUCE_SUM：求和。 此默认类型不可用，用户需配置此项参数。
axis	SVector<int64_t>	{}	指定轴（维度）。 ● axis不能为空且长度要求小于等于输入x的维度。 ● axis可以支持多个轴上进行处理，各元素要求小于x的维度且大于等于0。
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	● REDUCE_MAX/REDUCE_MIN：int32/bf16 ● REDUCE_SUM：float16/bf16	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
output	基于输入“x”的维度，消除axis要求的维度。	● REDUCE_MAX/REDUCE_MIN：int32/bf16 ● REDUCE_SUM：float16/bf16	ND	输出tensor。

 版权所有 © 华为技术有限公司

规格约束

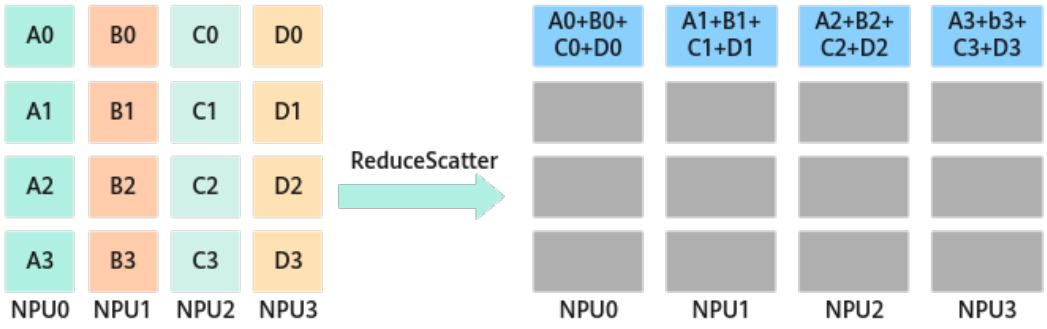
- axis不能为空且长度要求小于等于输入x的维度。
- axis中各元素要求小于x的维度且大于等于0。
- reduceType必须是REDUCE_MAX、REDUCE_MIN或REDUCE_SUM。

10.48 ReduceScatterOperation

功能

多对多。在集群内的所有节点上都按维度执行相同的Reduce规约运算，再将结果发散到集群内所有的节点上。

图 10-48 通信原理图



注：硬件类型仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品，不支持确定性计算和多机通信。

使用场景

- 可应用于数据并行和模型并行。
- 数据并行allReduce里的 ReduceScatter+ Allgather组合里的ReduceScatter操作。
- 模型并行里在前向allgather后的反向计算里的ReduceScatter。

使用示例

以rankSize=2为例

reduceType=sum

输入tensor的shape为（2 * 3 * 4）

输入tensor:

```
input_tensor_0 = [[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]],
                  [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]]
input_tensor_1 = [[[25, 26, 27, 28], [29, 30, 31, 32], [33, 34, 35, 36]],
                  [[37, 38, 39, 40], [41, 42, 43, 44], [45, 46, 47, 48]]]
```

输出tensor的shape为（1 * 3 * 4）

输出tensor:

```
output_tensor_0 = [[[26, 28, 30, 32],[34, 36, 38, 40],[42, 44, 46, 48]]]
output_tensor_1 = [[[50, 52, 54, 56],[58, 60, 62, 64],[66, 68, 70, 72]]]
```

 版权所有 © 华为技术有限公司

定义

```
struct ReduceScatterParam{
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    std::string reduceType = "sum";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string backend = "lccl";
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
rank	int	0	[0, rankSize-1]	是	当前卡所属通信编号。
rankSize	int	0	-	是	通信的卡的数量。
rankRoot	int	0	[0, rankSize-1]	是	主通信编号。
reduceType	string	"sum"	sum prod max min	是	通信计算类型。 支持“sum”（相加），“prod”（相乘），“max”（取最大）和“min”（取最小）。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
backend	string	“lccl”	lccl/hccl	是	通信计算类型，仅支持“hccl”和“lccl”。 ● “backend”为“hccl”时 – 支持"sum"，"prod"，"max"和"min"。 – Atlas 800I A2 推理产品/Atlas A2 训练系列产品上“reduceType”为"prod"时，不支持数据类型int16和bf16。 – Atlas 训练系列产品不支持int16、bf16。 ● “backend”为“lccl”时 – 支持"sum"，"max"和"min"。 – 机器为Atlas A2 训练系列产品里的Atlas 200T A2 Box16 异构子框时支持单机16卡（16卡全量拓扑通信或单节点内任意卡通信）。
hcclComm	HcclComm	nullptr	-	否	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域,则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	COMM_MULTI_PROCESS/COMM_MULTI_THREAD	否	通信模式，CommMode类型枚举值。
rankTableFile	string	无	-	否	集群信息的配置文件路径。
commDomain	string	无	-	否	通信device组用通信域名标识，多通信域时使用。当backend为lccl时，commMode为多进程时，commDomain需要设置0-63。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
rsv[64]	uint8_t	{0}	[0]	否	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[rankSize*n, dim_0, dim_1, ..., dim_n]	"lccl": float16/float/ int8/int16/ int32/bf16 "hccl": float16/float/ int8/int16/ int32/bf16	ND	输入tensor，维度小于等于8，第一维的大小为rankSize的n倍（n为正整数）。

输出

参数	维度	数据类型	格式	描述
output	[n, dim_0, dim_1, ..., dim_n]	"lccl": float16/float/ int8/int16/ int32/bf16 "hccl": float16/float/ int8/int16/ int32/bf16	ND	输出tensor，维度小于等于8，第一维的大小n=x[0]/rankSize，从第二维开始输出output的维数与输入x的维度以及维度值相等，数据类型也相同。

规格约束

- rank、rankSize、rankRoot需满足以下条件。
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：

```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```

- 当前不支持单device上跑多个HCCP进程实例，不能单卡同时跑多个通信算子。比如运行通信算子场景下，不支持同一张卡上跑多个模型。

 版权所有 © 华为技术有限公司

10.49 ReduceScatterVOperation

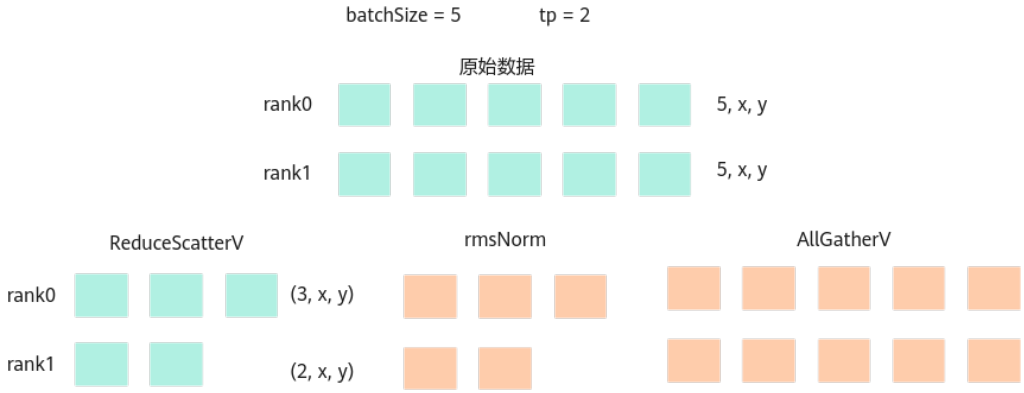
功能

将多个通信卡上的数据进行计算，支持相加、取最大、最小三种计算，然后不等地发送到每张卡上。

使用场景

推理场景中会出现batch size不能被TP数整除的情况，reducescatter后续的计算算子需要按照batch维度处理数据，再将处理数据进行allgather，如图图10-49所示。

图 10-49 算子上文示意图



对于原始的ReduceScatter来说，要求在每张卡的数据一样多，对于ReduceScatterV来说，假设有5份数据，可以在一张卡上分三份，一张卡上分两份。

使用示例：

- **输入**

```
[[[0], [1], [2], [3], [4]], [[0], [1], [2], [3], [4]]]
[[2, 3], [2, 3]]
[[0, 2], [0, 2]]
[2, 3]
```

- **输出**

  NPU0 的输出：

```
[[0], [2]]
```

  NPU1 的输出：

```
[[4], [6], [8]]
```

定义

```
struct ReduceScatterVParam {
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    std::vector<int64_t> sendCounts;
    std::vector<int64_t> sdispls;
    std::int64_t recvCount = 0;
    std::string reduceType = "sum";
    HcclComm hcclComm = nullptr;
```

 版权所有 © 华为技术有限公司

```
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string backend = "hccl";
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
rank	int	0	[0, rankSize-1]	是	当前卡所属通信编号。
rankSize	int	0	-	是	通信的卡的数量。
rankRoot	int	0	[0, rankSize-1]	是	主通信编号。
reduceType	string	"sum"	sum prod max min	是	通信计算类型。 支持“sum”（相加），“prod”（相乘），“max”（取最大）和“min”（取最小）。
backend	string	“hccl”	hccl	是	通信计算类型，仅支持“hccl”。 ● 支持"sum"，"prod"，"max"和"min"。 ● “reduceType”为"prod"时，不支持数据类型为int16和bf16。 ● Atlas 推理系列产品不支持int64、bf16，int16只有“reduceType”为"sum"时支持。
hcclComm	HcclComm	nullptr	-	否	HCCL通信域指针。当前算子仅支持lccl，此参数为预留参数。默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	COMM_MULTI_PROCESS/COMM_MULTI_THREAD	否	通信模式，CommMode类型枚举值。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
rankTableFile	string	无	-	否	集群信息的配置文件路径。
commDomain	string	无	-	否	通信device组用通信域名标识。
rsv[64]	uint8_t	{0}	[0]	否	预留参数。

输入

参数	维度	数据类型	格式	是否必选	描述
x	[dim_0, dim_1]	"hccl": float16/int8	ND	是	输入tensor，dim_0，dim_1大小没有限制。
sendCounts	[rankSize]	int64	ND	是	一维张量，其长度等于ranksize卡数，张量里的每个索引的值代表着每张卡分配的数据量，例如sendCounts[0]：rank0分配的数据量。张量中值的和等于x的dim_0。
sdispls	[rankSize]	int64	ND	是	一维张量 ，其长度等于ranksize卡数，张量里的每个索引的值代表着从对应索引卡号接收到的数据量的偏移，sdispls [0] = n表示rank0从相对于输入起始位置的偏移量为n的位置开始接收sendCounts[0]的数据量。张量中的值需小于x的dim_0。
recvCount	[1]	int64	ND	是	一维张量，该维度大小就是该固定1，recvCount 代表着该卡接收到的数据量。数值与sendCounts保持一致。
y	[dim_0]	float16	ND	是	一维张量，该张量用来推导output的shape。数值与recvCount相等。

输出

参数	维度	数据类型	格式	是否必选	描述
output	[dim_0, dim_1]	"hccl": float16/int8	ND	是	输出tensor，dim_0为y的shape。dim_0等于recvCount.shape[rank]，dim_1等于x.shape[1]。

 版权所有 © 华为技术有限公司

规格约束

- rank、rankSize、rankRoot需满足以下条件。
  - 0 ≤ rank < rankSize
  - 0 ≤ rankRoot < rankSize

接口调用示例（python）

```
def main_worker(rank, world_size,inTensorDtypes, sizes, random_seed):
    # init process group
    torch_npu.npu.set_device(rank)
    print(f'Process {rank} started, using device npu:{rank}.')
    # init reduce_scatterv_operation
    op_name = "ReduceScatterVOperation"
    reduce_scatterv_operation = torch.classes.OperationTorch.OperationTorch(
        "ReduceScatterVOperation")
    torch.manual_seed(random_seed)
    low = -100
    high = 100
    for inTensorDtype in inTensorDtypes:
        inTensors=[torch.tensor([[0],[1],[2],[3],[4]],dtype=inTensorDtype),torch.tensor([[0],[1],[2],[3],
[4]],dtype=inTensorDtype)]
        # y用来推导outputshape，ranksize为几长度就为几
        if rank == 0:
            y = torch.tensor([0, 1], dtype=inTensorDtype)
        else:
            y = torch.tensor([0,1,2], dtype=inTensorDtype)
        GoldenTensors=[torch.tensor([[0],[2]],dtype=inTensorDtype),torch.tensor([[4],[6],
[8]],dtype=inTensorDtype)]
        acl_param = json.dumps({"rank": rank, "rankSize": world_size, "sendCounts":sendcount[rank],
                                "sdispls":senddisp[rank], "recvCount":recvout[rank], "rankRoot": 0, "backend": "hccl"})
        run_param =
json.dumps({"sendCounts":sendcount[rank],"sdispls":senddisp[rank],"recvCount":recvout[rank]})
        host_list = [sendcount[rank],senddisp[rank],[recvout[rank]]]
        host_tensors =  [np.array(tensor) for tensor in host_list]
        host_tensors = [torch.from_numpy(tensor).to(torch.int64) for tensor in host_tensors]
        host_tensors = [tensor.npu() for tensor in host_tensors]
        reduce_scatterv_operation.set_param(acl_param)
        reduce_scatterv_operation.set_varaintpack_param(run_param)
        acl_out_tensor =
reduce_scatterv_operation.execute([inTensors[rank].npu(),host_tensors[0],host_tensors[1],host_tensors[2],
y.npu()])[0]
        print(f'acl_outtensor:{acl_out_tensor}')
        torch.npu.synchronize()
        # assert result
        assert golden_compare(acl_out_tensor.cpu(), GoldenTensors[rank])

class reduce_scatterv_operationTest(operation_test.OperationTest):
    def test_reduce_scatterv_operation(self):
        world_size = 2
        random_seed = 123
        inTensorDtypes = [torch.float16]
        sizes = [[3,4]]
```

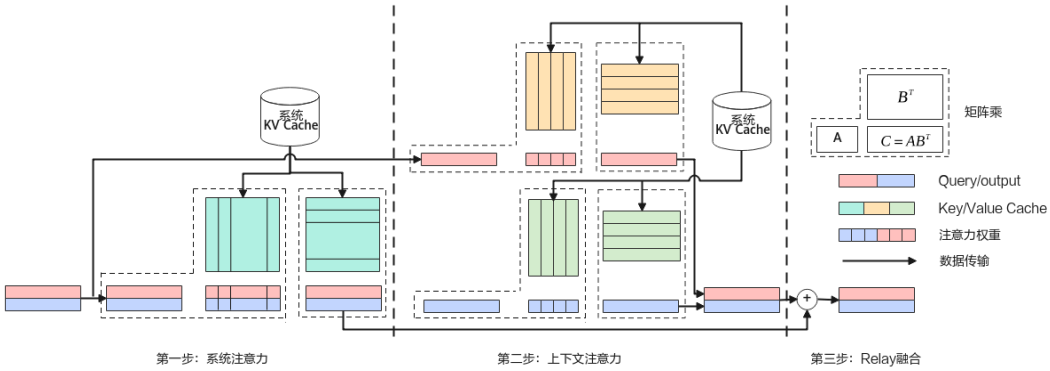
10.50 RelayAttentionOperation

功能

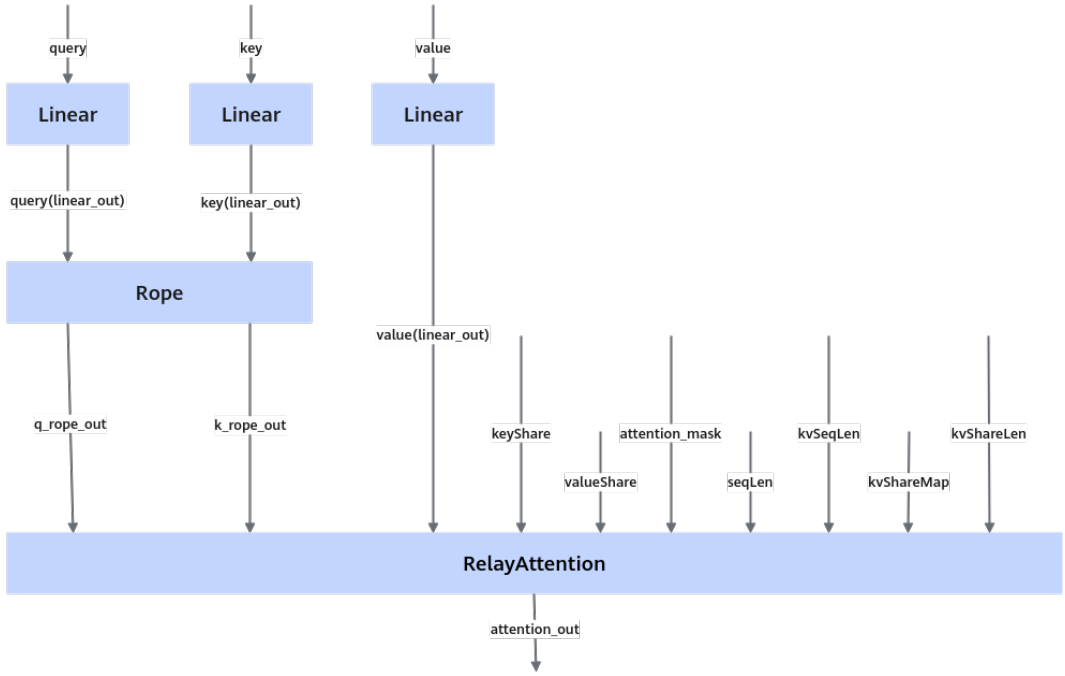
RelayAttention算法的核心创新点是通过将与系统提示对应的矩阵-向量乘法分组为矩阵-矩阵乘法，允许对于一批输入令牌从DRAM中仅读取一次系统提示的隐藏状态（键值对），从而消除现有因果注意力计算算法中处理系统提示时存在的大量冗余内存访问，在提高效率的同时保持生成质量且无需模型重新训练。

 版权所有 © 华为技术有限公司

图 10-50 RelayAttention 计算原理



算子上下文



定义

```
struct RelayAttentionParam {
    int32_t headNum = 0;
    float qkScale = 1;
    int32_t kvHeadNum = 0;
    enum MaskType : int {
        MASK_TYPE_UNDEFINED = 0,
        MASK_TYPE_NORM,
    };
    MaskType maskType = MASK_TYPE_UNDEFINED;
    uint8_t rsv[32] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	描述
headNum	int32_t	0	head数量。headNum需要大于0。
qkscale	float	1.0	算子tor值。
kvHeadNum	int32_t	0	kvhead数量。取值范围为[0, 8]。 kvHeadNum取值分为两种情况： ● 取值为零。 ● 取值大于0，此时kvHeadNum小于等于headNum且headNum为kvHeadNum的整数倍。
maskType	MaskType	MASK_TYPE_UNDEFINED	mask类型。
rsv[32]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	CPU or NPU	描述
query	[B, qHiddenSize]	float16/bf16	ND	NPU	query矩阵。
key	[B, [S1, N, D]] [B, [S1, N*D]]	float16/bf16	ND	CPU	不共享key矩阵，为TensorList，可选择是否合轴。
value	[B, [S1, N, D]] [B, [S1, N*D]]	float16/bf16	ND	CPU	不共享value矩阵，为TensorList，可选择是否合轴。
keyShare	[BS, [S2, N, D]] [BS, [S2, N*D]]	float16/bf16	ND	CPU	共享key矩阵，为TensorList，可选择是否合轴。
valueShare	[BS, [S2, N, D]] [BS, [S2, N*D]]	float16/bf16	ND	CPU	共享value矩阵，为TensorList，可选择是否合轴。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	CPU or NPU	描述
attentionMask	-	float16/bf16	ND	NPU	预留输入tensor。
seqLen	[B]	int32	ND	CPU	qseqlen list。
kvSeqLen	[B]	int32	ND	CPU	kvseqlen list。
kvShareMap	[B]	int32	ND	CPU	batch和共享组映射关系数组，无共享组时取值为-1。
kvShareLen	[BS]	int32	ND	CPU	共享组实际长度数组。

输出

参数	维度	数据类型	格式	CPU or NPU
out	[B, qHiddenSize]	float16/bf16	ND	NPU

规格说明

- 仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 输入key、value、keyShare、valueShare为TensorList，这些输入tensor中B、BS为二级指针维度。以key为例，其为大小为[B]的list，里面存储B个指针，每个指针指向一个[S1, N ,D]（或[S1, N *D]）大小的tensor。
- B为batch，取值范围为(0, 60]，BS为共享组个数，S1为不共享的长度，S2为共享长度，N为kvhead，D为head_size。
- CPU是指该输入tensor为HostTensor，传入算子时需要bind tensor，而NPU则为device tensor。
- key和value应同时选择合轴或者不合轴，keyShare和valueShare应同时选择合轴或者不合轴。即key和value维度相等，keyShare和valueShare维度相等。
- 数据类型同时支持float16/bf16的输入tensor，必须同时为float16或同时为bf16。
- maskType当前只支持MASK_TYPE_UNDEFINED，故attentionMask为预留接口，必须传入且仅限制其数据类型。

10.51 RepeatOperation

功能

将输入tensor的Shape，根据参数中multiples上的信息扩展指定倍数。

 版权所有 © 华为技术有限公司

硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 200I/500 A2 推理产品	支持	不支持输入输出Tensor数据类型为bf16的场景。
Atlas 推理系列产品	支持	不支持输入输出tensor数据类型为bf16的场景。
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

定义

```
struct RepeatParam {
    SVector<int64_t> multiples;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
multiples	SVector<int64_t>	-	输入tensor每个维度上扩展的倍数，支持对输入tensor的广播。 支持在不超过两个维度上进行扩展。 multiples的维度应小于等于8，且大于或等于输入tensor的维度，每一个元素要求都大于0。
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[x_dim_0，x_dim_1，... ，x_dim_n]	float16/bf16	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
y	[y_dim_0，y_dim_1，... ，y_dim_n]	float16/bf16	ND	输出tensor。和输入的数据类型一致。

 版权所有 © 华为技术有限公司

规格约束

- 输出y的维度和multiples维度一致，每个维度大小为输入x广播到multiples维度后和multiples对应维度的乘积。
- 输入tensor中shape大于1且需要扩展的维度数加输入tensor的维度数需要小于等于8。

功能列表

- **当multiples的长度与输入tensor的维度相同时**

  输出tensor的每个维度大小为输入tensor的相应维度大小与参数multiples对应位置的乘积。

  例如输入一个Tensor的shape为[1, 2, 3]，参数multiples为{1, 2, 2}。此时的输出tensor的shape为[1, 4, 6]。

  - inTensor:

    ```
    [[[1,2,3]
      [4,5,6]]]
    ```

    通过repeat之后为

  - outTensor:

    ```
    [[[1,2,3,1,2,3],
      [4,5,6,4,5,6],
      [1,2,3,1,2,3],
      [4,5,6,4,5,6]]]
    ```

- **当multiples的长度大于输入tensor的维度时**

  输出tensor的每个维度大小为输入tensor维度广播到参数multiples长度后与其对应位置的乘积。

  例如输入一个Tensor的shape为[2, 3]，参数multiples为{1, 2, 2}。此时的输出tensor的shape为[1, 4, 6]。

  - inTensor:

    ```
    [[1,2,3]
     [4,5,6]]
    ```

    通过repeat之后为

  - outTensor:

    ```
    [[[1,2,3,1,2,3],
      [4,5,6,4,5,6],
      [1,2,3,1,2,3],
      [4,5,6,4,5,6]]]
    ```

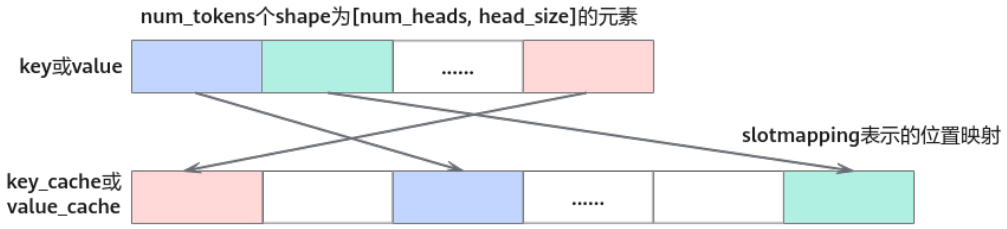
10.52 ReshapeAndCacheOperation

10.52.1 功能

遍历每个key和value，将key和value(num_heads, head_size)以slotmapping填入key_cache/value_cache指定位置。

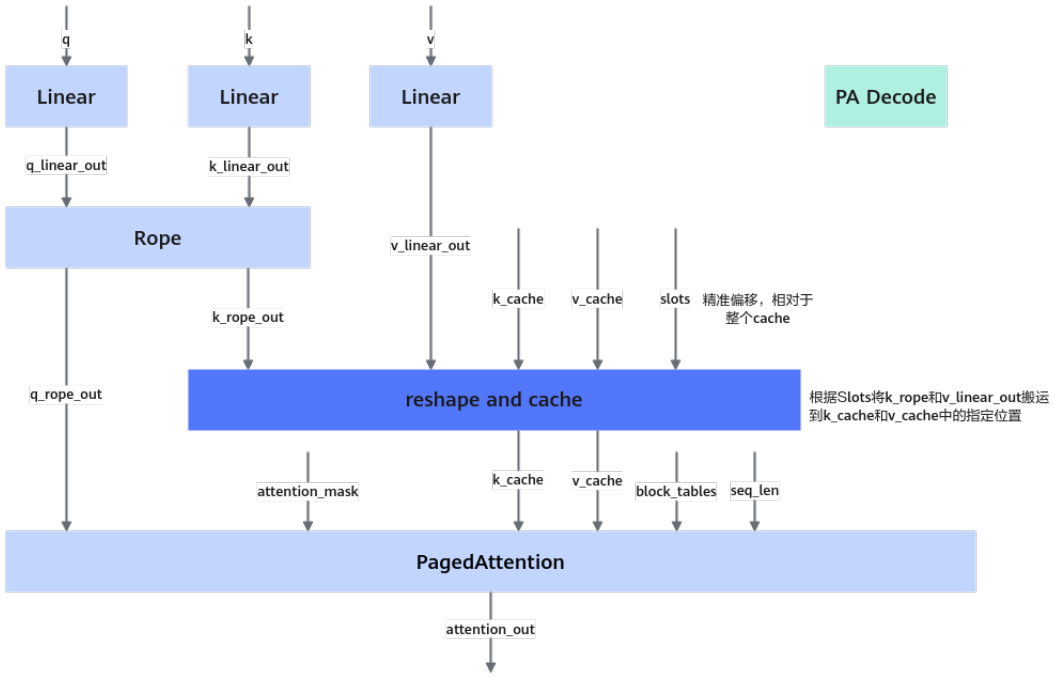
 版权所有 © 华为技术有限公司

图 10-51 ReshapeAndCacheOperation 功能示意图



10.52.2 算子上下文

图 10-52 ReshapeAndCacheOperation 算子上下文



如上图所示，为典型rope场景下pa decoder layer的部分结构。ReshapeAndCache与paged attention配合使用，用于存储某layer上各个token的历史key和value。与加速库的KVCache算子不同，ReshapeAndCache算子只维护某一层的kvcache而非一个服务的全部kvcache。因此，若模型中有n个layer，使用ReshapeAndCache的情况下需要n个k_cache tensor与n个v_cache tensor管理历史kv。

10.52.3 定义

```
struct ReshapeAndCacheParam {
  enum CompressType : int {
    COMPRESS_TYPE_UNDEFINED = 0,
    COMPRESS_TYPE_KVHEAD,
    COMPRESS_TYPE_KVHEAD_ROPE
  };
  enum KvCacheCfg : int {
    K_CACHE_V_CACHE = 0,
    K_CACHE_V_BYPASS,
    K_CACHE_V_CACHE_NZ
  };
  CompressType compressType = COMPRESS_TYPE_UNDEFINED;
  KvCacheCfg kvCacheCfg = K_CACHE_V_CACHE;
```

 版权所有 © 华为技术有限公司

```
    uint8_t rsv[16] = {0};
};
```

10.52.4 参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
compressType	CompressType	COMPRESS_TYPE_UNDEFINED	0，1，2	否	压缩类型。 • COMPRESS_TYPE_UNDEFINED：默认值，不压缩。 • COMPRESS_TYPE_KVHEAD：压缩key_cache, value_cache的kvHead维度。 • COMPRESS_TYPE_KVHEAD_ROPE：rope场景下压缩key_cache，value_cache的kvHead维度。
kvCacheCfg	KvCacheCfg	K_CACHE_V_CACHE	0，1，2	否	KvCache配置。 • K_CACHE_V_CACHE：默认值，传入key_cache和value_cache。 • K_CACHE_V_BYPASS：只传入key_cache。 • K_CACHE_V_CACHE_NZ：传入key_cache和value_cache，且为NZ格式。
rsv[16]	uint8_t	{0}	[0]	否	预留参数。

10.52.5 输入输出

10.52.5.1 不开启多头压缩功能

compressType=COMPRESS_TYPE_UNDEFINED

输入

参数	维度	数据类型	格式	描述
key	[num_tokens, num_head, k_head_size]	float16/bf16/int8	ND	当前step多个token的key，不开启多头压缩功能时支持key和value最后一维不同。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
value	[num_tokens, num_head, v_head_size]	float16/bf16/int8	ND	当前step多个token的value，不开启多头压缩功能时支持key和value最后一维不同。
keyCache	[num_blocks, block_size, num_head, k_head_size]	float16/bf16/int8	ND/NZ	当前layer的key cache，不开启多头压缩功能时支持keyCache和valueCache最后一维不同。
valueCache	[num_blocks, block_size, num_head, v_head_size]	float16/bf16/int8	ND/NZ	当前layer的value cache，不开启多头压缩功能时支持keyCache和valueCache最后一维不同。
slotMapping	[num_tokens]	int32	ND	每个token key或value在cache中的存储偏移，即（block_id * block_size + offset_in_block），值域范围为(INT32_MIN, num_blocks * block_size)且不存在重复数值。取值为负数时不进行任何搬运操作。

输出

参数	维度	数据类型	格式	描述
keyCacheOut	[num_blocks, block_size, num_head, k_head_size]	float16/bf16/int8	ND/NZ	所有的key cache。与输入keyCache为同一地址。
valueCacheOut	[num_blocks, block_size, num_head, v_head_size]	float16/bf16/int8	ND/NZ	所有的value cache。与输入valueCache为同一地址。

10.52.5.2 开启 alibi 场景下的多头压缩功能

compressType=COMPRESS_TYPE_KVHEAD

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
key	[num_tokens, num_head, head_size]	float16/bf16/int8	ND	当前step多个token的key。最后一维需要是32的倍数。
value	[num_tokens, num_head, head_size]	float16/bf16/int8	ND	当前step多个token的value。最后一维需要是32的倍数。
keyCache	[num_blocks, block_size, 1, head_size]	float16/bf16/int8	ND	所有的key cache。
valueCache	[num_blocks, block_size, 1, head_size]	float16/bf16/int8	ND	所有的value cache。
slotMapping	[batch*num_head]	int32	ND	• 每个head每个batch的key或value在cache中的存储偏移，即（block_id * block_size + offset_in_block）。 • 取值范围为[0, num_blocks * block_size)，且不能存在重复值。即意味着num_tokens <= num_blocks*block_size。
wins	[batch*num_head]	int32	ND	压缩量。wins[i] < seqLens[floor(i/num_head]，且wins内的值需要大于等于0，值为0时不压缩。
seqLens	[batch]	int32	ND	每个batch的实际seqLen，内部值都大于0。

输出

参数	维度	数据类型	格式	描述
keyCacheOut	[num_blocks, block_size, 1, head_size]	float16/bf16/int8	ND	所有的key cache。与输入keyCache为同一地址。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
valueCacheOut	[num_blocks, block_size, 1, head_size]	float16/bf16/int8	ND	所有的value cache。与输入valueCache为同一地址。

10.52.5.3 开启 rope 场景下的多头压缩功能

compressType=COMPRESS_TYPE_KVHEAD_ROPE

输入

参数	维度	数据类型	格式	描述
key	[num_tokens, num_head, head_size]	float16/bf16	ND	当前step多个token的key。最后一维需要是32的倍数。
value	[num_tokens, num_head, head_size]	float16/bf16	ND	当前step多个token的value。最后一维需要是32的倍数。
keyCache	[num_blocks, block_size, 1, head_size]	float16/bf16	ND	所有的key cache。
valueCache	[num_blocks, block_size, 1, head_size]	float16/bf16	ND	所有的value cache。
slotMapping	[batch*num_head]	int32	ND	• 每个head每个batch的key或value在cache中的存储偏移，即（block_id * block_size + offset_in_block）。 • 取值范围为[0, num_blocks * kv_heads * block_size)，且不能存在重复值。即意味着num_tokens <= num_blocks*block_size。
wins	[batch*num_head]	int32	ND	压缩量。wins[i] < seqLens[floor(i/num_head]，且wins内的值需要大于等于0，值为0时不压缩。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
seqLens	[batch]	int32	ND	每个batch的实际seqLen，内部值都大于0。
offsetIndex	[batch*num_head]	int32	ND	每个batch每个head的压缩起点。取值范围为[-1, seqLens[i]-wins[i]]；取值为-1时不进行压缩操作，wins的值无要求。

输出

参数	维度	数据类型	格式	描述
keyCacheOut	[num_blocks, block_size, 1, head_size]	float16/bf16	ND	所有的key cache。与输入keyCache为同一地址。
valueCacheOut	[num_blocks, block_size, 1, head_size]	float16/bf16	ND	所有的value cache。与输入valueCache为同一地址。

10.52.5.4 开启单入单出功能

kvCacheCfg=K_CACHE_V_BYPASS，compressType=COMPRESS_TYPE_UNDEFINED

输入

参数	维度	数据类型	格式	描述
key	[num_tokens, k_num_head, k_head_size]	float16/bf16/int8	ND	当前step多个token的key。
keyCache	[num_blocks, block_size, k_num_head, k_head_size]	float16/bf16/int8	ND	当前layer的key cache。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
slotMapping	[num_tokens]	int32	ND	每个token key在cache中的存储偏移，即（block_id * block_size + offset_in_block），值域范围为(INT32_MIN, num_blocks * block_size)且不存在重复数值。取值为负数时不进行任何搬运操作。

输出

参数	维度	数据类型	格式	描述
keyCacheOut	[num_blocks, block_size, k_num_head, k_head_size]	float16/bf16/int8	ND	所有的key cache。与输入keyCache为同一地址。

10.52.5.5 开启支持输入转 NZ 格式功能

kvCacheCfg=K_CACHE_V_CACHE_NZ，
compressType=COMPRESS_TYPE_UNDEFINED

输入

参数	维度	数据类型	格式	描述
key	[num_tokens, num_head, head_size_k]	int8/float16/bf16	ND	head_size_k*sizeof(dtype_k)满足32字节对齐，同时小于98304。
value	[num_tokens, num_head, head_size_v]	float16/bf16	ND	head_size_v*sizeof(dtype_v)满足32字节对齐，同时小于98304。key支持int8时，value可以是float16或者bf16。
keyCache	[num_blocks, num_head*k_head_size // last_dim_k, block_size, last_dim_k]	int8/float16/bf16	NZ	当数据类型为int8时，last_dim_k为32，当数据类型是float16/bf16时，last_dim_k为16。数据类型与key保持一致。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
valueCache	[num_blocks, num_head*k_head_size // last_dim_v, block_size, last_dim_v]	float16/bf16	NZ	last_dim_v只能是16。数据类型与value保持一致。
slotMapping	[num_tokens]	int32	ND	last_dim_v只能是16。值域范围为(INT32_MIN, num_blocks * block_size)且不存在重复数值。取值为负数时不进行任何搬运操作。

输出

参数	维度	数据类型	格式	描述
keyCacheOut	[num_blocks, num_head*k_head_size // last_dim_k, block_size, last_dim_k]	int8/float16/bf16	NZ	当数据类型为int8时，last_dim_k为32，当数据类型是float16/bf16时，last_dim_k为16。
valueCacheOut	[num_blocks, num_head*k_head_size // last_dim_v, block_size, last_dim_v]	float16/bf16	NZ	last_dim_v只能是16。

10.52.6 规格约束

- ntokens = sum(seqlen[i])，i=0、1...batch-1。
- Atlas 推理系列产品上不支持压缩功能，且数据格式标明有NZ的需为NZ格式，keyCache和valueCache的维度需为[num_blocks, num_head*head_size/16, block_size, 16]，其中最后一维必须为16，block_size需16对齐。kvCacheCfg配置为K_CACHE_V_CACHE或K_CACHE_V_CACHE_NZ时，数据格式均为NZ。
- 在key和value最后一维不同的情况下只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品，此场景下keyCache、valueCache、keyCacheOut、valueCacheOut只支持ND格式。
- key、value、keyCache、valueCache、keyCacheOut 、valueCacheOut的数据类型都必须完全一致。
- keyCache、valueCache、和keyCacheOut 、valueCacheOut的数据格式必须一致。

 版权所有 © 华为技术有限公司

- 单入单出功能只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品，且不支持压缩场景。
- 压缩场景下cache的大小要大于压缩后的大小。
- 不开启多头压缩功能场景下，在Atlas 200I/500 A2 推理产品上NZ格式下仅支持float16数据类型。
- 开启支持输入转NZ格式功能场景下，有如下约束：
  - 只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
  - 支持key、value数据不等长功能，即key、value最后一维可以不相等。
  - key、value的数据类型可以不相同。

10.52.7 算子调用示例（C++）

10.52.7.1 reshape_and_cache_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"
#include <random>
#include <algorithm>
#include <unordered_set>

#include "demo_util.h"

uint32_t NUM_TOKENS = 2;
uint32_t NUM_HEAD = 32;
uint32_t K_HEAD_SIZE = 128;
uint32_t V_HEAD_SIZE = K_HEAD_SIZE;
uint32_t NUM_BLOCKS = 512;
uint32_t BLOCK_SIZE = 128;

/**
 * @brief 准备随机输入tensorK或输入tensorV的内容
 * @param kvflag 0:key; 1:value
 * @return 输入tensorK或输入tensorV
 */
std::vector<float> KvGeneration(bool kvflag)
{
    // 创建随机数生成器
    std::random_device rd;
    std::mt19937 gen(rd());
    // 定义随机数分布范围
    std::uniform_real_distribution<> dis(-100.0, 100.0);
    // 定义要生成的随机数的个数
    size_t num_elements = kvflag ? NUM_TOKENS * NUM_HEAD * V_HEAD_SIZE : NUM_TOKENS *
NUM_HEAD * K_HEAD_SIZE;
    // 创建一个 vector 并填充随机数
    std::vector<float> intensorKV;
    for (size_t i = 0; i < num_elements; ++i) {
        intensorKV.push_back(dis(gen));
    }
    return intensorKV;
}
```

 版权所有 © 华为技术有限公司

```
/**
 * @brief 准备随机输入tensorSlotmapping的内容
 * @param slotRange 数据生成范围
 * @param num_tokens Slotmapping length
 * @return 输入tensorSlotmapping
 */
std::vector<int32_t> SlotmappingGeneration(int slotRange, size_t num_tokens) {
    // 创建一个包含范围 [-slotRange, slotRange] 的所有整数的 vector
    std::vector<int32_t> all_numbers;
    for (int i = -slotRange; i <= slotRange; ++i) {
        all_numbers.push_back(i);
    }
    // 检查 num_tokens 是否超过范围内的整数数量
    if (num_tokens > all_numbers.size()) {
        throw std::invalid_argument("num_tokens exceeds the range of unique numbers available");
    }
    // 创建随机数生成器
    std::random_device rd;
    std::mt19937 gen(rd());
    // 打乱所有整数
    std::shuffle(all_numbers.begin(), all_numbers.end(), gen);
    // 选取前 num_tokens 个元素
    std::vector<int32_t> slotmapping(all_numbers.begin(), all_numbers.begin() + num_tokens);
    return slotmapping;
}

/**
 * @brief 准备atb::VariantPack
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。序列长度向量，等于1时，为增量或全量；大于1时，为全量
 * @param tokenOffsetHost host侧tensor。计算完成后的token偏移
 * @param layerId layerId，取cache的kv中哪一个kv进行计算
 * @return atb::VariantPack
 */
atb::VariantPack PrepareVariantPack(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建key，value tensor
    std::vector<float> keyData = KvGeneration(false);
    atb::Tensor tensorKey = CreateTensorFromVector(
        contextPtr, stream, keyData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NUM_TOKENS,
NUM_HEAD, K_HEAD_SIZE});
    std::vector<float> valueData = KvGeneration(true);
    atb::Tensor tensorValue = CreateTensorFromVector(
        contextPtr, stream, valueData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NUM_TOKENS,
NUM_HEAD, V_HEAD_SIZE});
    // 创建kvCache tensor
    std::vector<float> kCacheData(NUM_BLOCKS * BLOCK_SIZE * NUM_HEAD * K_HEAD_SIZE, 0);
    atb::Tensor tensorKCache = CreateTensorFromVector(
        contextPtr, stream, kCacheData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NUM_BLOCKS,
BLOCK_SIZE, NUM_HEAD, K_HEAD_SIZE});
    std::vector<float> vCacheData(NUM_BLOCKS * BLOCK_SIZE * NUM_HEAD * V_HEAD_SIZE, 0);
    atb::Tensor tensorVCache = CreateTensorFromVector(
        contextPtr, stream, vCacheData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NUM_BLOCKS,
BLOCK_SIZE, NUM_HEAD, V_HEAD_SIZE});
    // 创建SlotMapping
    std::vector<int32_t> slotMappingData = SlotmappingGeneration(NUM_BLOCKS * BLOCK_SIZE,
NUM_TOKENS);
    atb::Tensor tensorSlotMapping = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND,
{NUM_TOKENS});
    CHECK_STATUS(aclrtMemcpy(tensorSlotMapping.deviceData,
        tensorSlotMapping.dataSize,
        slotMappingData.data(),
        sizeof(int32_t) * slotMappingData.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {tensorKey,
        tensorValue,
```

 版权所有 © 华为技术有限公司

```
        tensorKCache,
        tensorVCache,
        tensorSlotMapping};
    // 准备输入张量
    atb::VariantPack variantPack;
    variantPack.inTensors = inTensors;  // 放入输入tensor
    variantPack.outTensors = {tensorKCache, tensorVCache};  // 放入输出tensor
    return variantPack;
}

/**
 * @brief 创建一个Reshape and Cache的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::ReshapeAndCacheParam opParam;
    opParam.compressType =
atb::infer::ReshapeAndCacheParam::CompressType::COMPRESS_TYPE_UNDEFINED;
    opParam.kvCacheCfg = atb::infer::ReshapeAndCacheParam::KvCacheCfg::K_CACHE_V_CACHE;
    atb::Operation *reshapeAndCacheOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(opParam, &reshapeAndCacheOp));
    return reshapeAndCacheOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // RAC示例
    atb::Operation *reshapeAndCacheOp = PrepareOperation();
    // 准备variantPack
    atb::VariantPack variantPack = PrepareVariantPack(context, stream);
    uint64_t workspaceSize = 0;
    // 对输入tensor和输出tensor进行校验
    CHECK_STATUS(reshapeAndCacheOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // RAC执行
    reshapeAndCacheOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(reshapeAndCacheOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "Reshape and Cache demo success!" << std::endl;
    return 0;
}
```

 版权所有 © 华为技术有限公司

10.52.7.2 reshape_and_cache_inference_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 推理系列产品。

与示例1相较本示例主要有以下修改点：

- kvCache shape改为：[block_num, head_size * head_num / 16, block_size, 16]。
- kvCache数据格式改为：ACL_FORMAT_FRACTAL_NZ。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"
#include <random>
#include <algorithm>
#include <unordered_set>

#include "demo_util.h"

uint32_t NUM_TOKENS = 3;
uint32_t NUM_HEAD = 4;
uint32_t K_HEAD_SIZE = 128;
uint32_t V_HEAD_SIZE = K_HEAD_SIZE;
uint32_t NUM_BLOCKS = 512;
uint32_t BLOCK_SIZE = 128;

/**
 * @brief 准备随机输入tensorK或输入tensorV的内容
 * @param kvflag 0:key; 1:value
 * @return 输入tensorK或输入tensorV
 */
std::vector<int8_t> KvGeneration(bool kvflag)
{
    // 创建随机数生成器
    std::random_device rd;
    std::mt19937 gen(rd());
    // 定义随机数分布范围
    std::uniform_real_distribution<> dis(-100.0, 100.0);
    // 定义要生成的随机数的个数
    size_t num_elements = kvflag ? NUM_TOKENS * NUM_HEAD * V_HEAD_SIZE : NUM_TOKENS *
NUM_HEAD * K_HEAD_SIZE;
    // 创建一个 vector 并填充随机数
    std::vector<int8_t> intensorKV;
    for (size_t i = 0; i < num_elements; ++i) {
        intensorKV.push_back(dis(gen));
    }
    return intensorKV;
}

/**
 * @brief 准备随机输入tensorSlotmapping的内容
 * @param slotRange 数据生成范围
 * @param num_tokens Slotmapping length
 * @return 输入tensorSlotmapping
 */
std::vector<int32_t> SlotmappingGeneration(int slotRange, size_t num_tokens)
{
    // 创建一个包含范围 [-slotRange, slotRange] 的所有整数的 vector
    std::vector<int32_t> all_numbers;
    for (int i = -slotRange; i <= slotRange; ++i) {
        all_numbers.push_back(i);
    }
```

 版权所有 © 华为技术有限公司

```
    // 检查 num_tokens 是否超过范围内的整数数量
    if (num_tokens > all_numbers.size()) {
        throw std::invalid_argument("num_tokens exceeds the range of unique numbers available");
    }
    // 创建随机数生成器
    std::random_device rd;
    std::mt19937 gen(rd());
    // 打乱所有整数
    std::shuffle(all_numbers.begin(), all_numbers.end(), gen);
    // 选取前 num_tokens 个元素
    std::vector<int32_t> slotmapping(all_numbers.begin(), all_numbers.begin() + num_tokens);
    return slotmapping;
}

/**
 * @brief 准备atb::VariantPack
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。序列长度向量，等于1时，为增量或全量；大于1时，为全量
 * @param tokenOffsetHost host侧tensor。计算完成后的token偏移
 * @param layerId layerId，取cache的kv中哪一个kv进行计算
 * @return atb::VariantPack
 */
atb::VariantPack PrepareVariantPack(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建key，value tensor
    std::vector<int8_t> keyData = KvGeneration(false);
    atb::Tensor tensorKey = CreateTensorFromVector(
        contextPtr, stream, keyData, ACL_INT8, aclFormat::ACL_FORMAT_ND, {NUM_TOKENS, NUM_HEAD,
K_HEAD_SIZE});
    std::vector<int8_t> valueData = KvGeneration(true);
    atb::Tensor tensorValue = CreateTensorFromVector(
        contextPtr, stream, valueData, ACL_INT8, aclFormat::ACL_FORMAT_ND, {NUM_TOKENS, NUM_HEAD,
V_HEAD_SIZE});
    // 创建kvCache tensor
    std::vector<int8_t> kCacheData(NUM_BLOCKS * BLOCK_SIZE * NUM_HEAD * K_HEAD_SIZE, 0);
    atb::Tensor tensorKCache = CreateTensorFromVector(contextPtr,
        stream,
        kCacheData,
        ACL_INT8,
        aclFormat::ACL_FORMAT_FRACTAL_NZ,
        {NUM_BLOCKS, NUM_HEAD * K_HEAD_SIZE / 16, BLOCK_SIZE, 16});
    std::vector<int8_t> vCacheData(NUM_BLOCKS * BLOCK_SIZE * NUM_HEAD * V_HEAD_SIZE, 0);
    atb::Tensor tensorVCache = CreateTensorFromVector(contextPtr,
        stream,
        vCacheData,
        ACL_INT8,
        aclFormat::ACL_FORMAT_FRACTAL_NZ,
        {NUM_BLOCKS, NUM_HEAD * V_HEAD_SIZE / 16, BLOCK_SIZE, 16});
    // 创建SlotMapping
    std::vector<int32_t> slotMappingData = SlotmappingGeneration(NUM_BLOCKS * BLOCK_SIZE,
NUM_TOKENS);
    atb::Tensor tensorSlotMapping = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND,
{NUM_TOKENS});
    CHECK_STATUS(aclrtMemcpy(tensorSlotMapping.deviceData,
        tensorSlotMapping.dataSize,
        slotMappingData.data(),
        sizeof(int32_t) * slotMappingData.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {tensorKey, tensorValue, tensorKCache, tensorVCache,
tensorSlotMapping};
    // 准备输入张量
    atb::VariantPack variantPack;
    variantPack.inTensors = inTensors;                    // 放入输入tensor
    variantPack.outTensors = {tensorKCache, tensorVCache};  // 放入输出tensor
    return variantPack;
}
```

 版权所有 © 华为技术有限公司

```
/**
 * @brief 创建一个Reshape and Cache的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::ReshapeAndCacheParam opParam;
    opParam.compressType =
atb::infer::ReshapeAndCacheParam::CompressType::COMPRESS_TYPE_UNDEFINED;
    opParam.kvCacheCfg = atb::infer::ReshapeAndCacheParam::KvCacheCfg::K_CACHE_V_CACHE;
    atb::Operation *reshapeAndCacheOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(opParam, &reshapeAndCacheOp));
    return reshapeAndCacheOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // RAC示例
    atb::Operation *reshapeAndCacheOp = PrepareOperation();
    // 准备variantPack
    atb::VariantPack variantPack = PrepareVariantPack(context, stream);
    uint64_t workspaceSize = 0;
    // 对输入tensor和输出tensor进行校验
    CHECK_STATUS(reshapeAndCacheOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // RAC执行
    reshapeAndCacheOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream)); // 流同步，等待device侧任务计算完成
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(reshapeAndCacheOp)); // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context)); // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "Reshape and Cache demo success!" << std::endl;
    return 0;
}
```

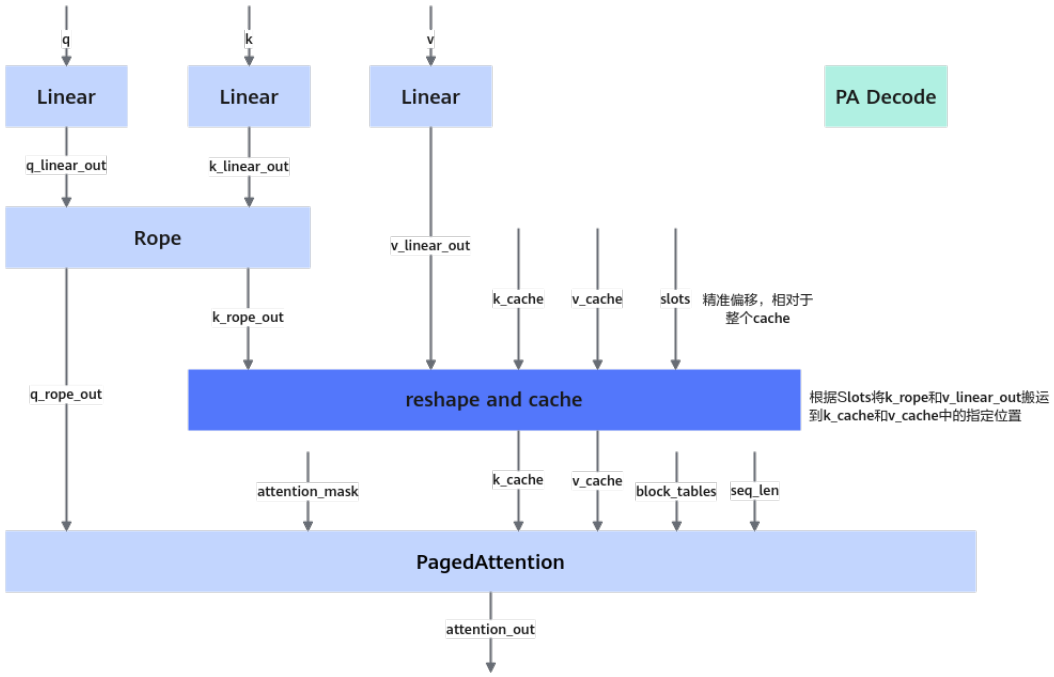
10.53 ReshapeAndCacheOmniOperation

功能

OmniAttention在Decode阶段通过对LLM特定的Layer的特定Head进行Attention稀疏计算达到节省KV Cache显存、降低片上内存搬运数据量到缓存、减少Attention计算量的性能增益。

 版权所有 © 华为技术有限公司

算子上下文



定义

```
struct ReshapeAndCacheOmniParam{
    uint8_t rsv[8] = {0};
}
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
rsv[8]	uint8_t	{0}	[0]	否	预留参数。

Omni Head 压缩功能

- **功能描述**
  - Omni Head：进行压缩，KV Cache只需要存储（First Sink Size+Recent Sink Size）个Token，不需要完整Seqlen个Token。
  - Full Head：不进行压缩。

  在模型中，特定的头采用omni压缩，对应使用本算子；其他头不压缩，使用原版ReshapeAndCache算子。
- **硬件支持情况**

硬件型号	支持情况	特殊说明
Atlas 推理系列产品	不支持	-

 版权所有 © 华为技术有限公司

硬件型号	支持情况	特殊说明
Atlas 800I A2 推理产品	支持	-

- **输入**

参数	维度	数据类型	格式	描述
key	[num_tokens, num_head, head_size]	float16/ bf16	ND	待被cache的key。head_size需要为32的倍数。
value	[num_tokens, num_head, head_size]	float16/ bf16	ND	待被cache的value。
key_cache	[num_blocks, block_size, 1, head_size	float16/ bf16	ND	被压缩后的cache好的历次key。
value_cache	[num_blocks, block_size, 1, head_size]	float16/ bf16	ND	被压缩后的cache好的历次value。
slot_mapping	[batch * num_head]	int32	ND	每个batch每个head的token key或value在cache中的存储偏移，即（block_id * block_size + offset_in_block）；取值范围为[0, num_blocks * block_size)，且不能存在重复值。
wins	[batch * num_head]	int32	ND	压缩量。wins[i] < seqLens[floor(i/num_head)]，且wins内的值需要大于等于0，值为0时不压缩。
seqLen	[batch]	int32	ND	每个batch的实际seqLen，内部值都大于0。
offsetIndex	[batch * num_head]	int32	ND	每个batch每个head的压缩起点。取值范围为[-1, seqLens[floor(i/num_head)]-wins[i]]；取值为-1时不进行压缩操作，wins的值无要求。

- **输出**

参数	维度	数据类型	格式	描述
key_cache	[num_blocks, block_size, 1, head_size]	float16/ bf16	ND	刷入key的key_cache。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
value_cache	[num_blocks, block_size, 1, head_size]	float16/bf16	ND	刷入value的value_cache。

- **规格约束**
  - $$ntokens = \sum seqLen[i]$$
  - batch * num_head<500
  - head_size需要为32的倍数。
  - key、value、keyCache、valueCache、keyCacheOut 、valueCacheOut的数据类型都必须完全一致。

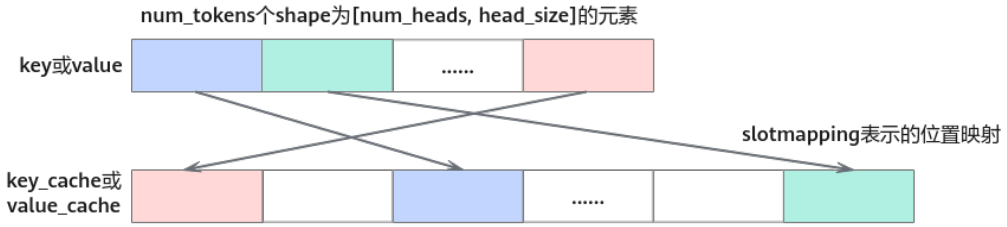
10.54 ReshapeAndCacheWithStrideOperation

10.54.1 功能

支持Leading Dimension功能，key和value由传入tensor：kStrides、vStrides、kOffset、vOffset确定。kStrides和vStrides表示从当前元素到下一个元素在内存中的跨度，kOffset表示key中第一个元素的位置相对于输入首地址的偏移，vOffset表示key中第一个元素的位置相对于输入首地址的偏移。

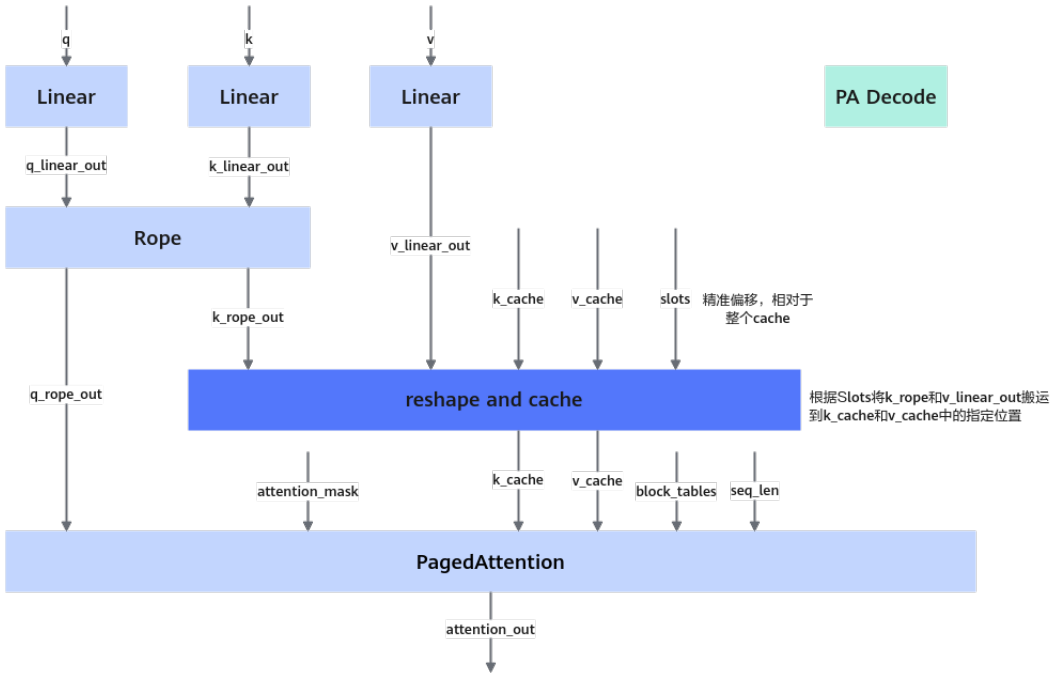
遍历每个key和value，将key和value(num_heads, head_size)以slotmapping填入key_cache/value_cache指定位置。

图 10-53 ReshapeAndCacheWithStrideOperation 功能示意图



 版权所有 © 华为技术有限公司

10.54.2 算子上下文



10.54.3 定义

```
struct ReshapeAndCacheWithStrideParam {
  enum CompressType : int {
    COMPRESS_TYPE_UNDEFINED = 0,
    COMPRESS_TYPE_KVHEAD,
    COMPRESS_TYPE_KVHEAD_ROPE
  };
  enum KvCacheCfg : int {
    K_CACHE_V_CACHE = 0,
    K_CACHE_V_BYPASS,
  };
  CompressType compressType = COMPRESS_TYPE_UNDEFINED;
  KvCacheCfg kvCacheCfg = K_CACHE_V_CACHE;
  uint8_t rsv[16] = {0};
};
```

 版权所有 © 华为技术有限公司

10.54.4 参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
compressType	CompressType	COMPRESS_TYPE_UNDEFINED	COMPRESS_TYPE_UNDEFINED：0	是	压缩类型。 • COMPRESS_TYPE_UNDEFINED：默认值，不压缩。 • COMPRESS_TYPE_KVHEAD：alibi场景下压缩key_cache，value_cahe的kvHead维度。 • COMPRESS_TYPE_KVHEAD_ROPE：rope场景下压缩key_cache, value_cahe的kvHead维度。
kvCacheCfg	KvCacheCfg	K_CACHE_V_CACHE	• K_CACHE_V_CACHE：0 • K_CACHE_V_BYPASS：1	是	KvCache配置。 • K_CACHE_V_CACHE：默认值，传入key_cache和value_cache。 • K_CACHE_V_BYPASS：只传入key_cache。
rsv[16]	uint8_t	{0}	[0]	否	预留参数。

10.54.5 输入输出

10.54.5.1 不开启多头压缩功能

compressType=COMPRESS_TYPE_UNDEFINED

输入

参数	维度	数据类型	格式	描述
key	[num_tokens, num_head, k_head_size]	float16/bf16	ND	当前step多个token的key，不开启多头压缩功能时支持key和value最后一维不同。
value	[num_tokens, num_head, v_head_size]	float16/bf16	ND	当前step多个token的value，不开启多头压缩功能时支持key和value最后一维不同。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
keyCache	[num_blocks, block_size, num_head, k_head_size]	float16/ bf16	ND	当前layer的key cache，不开启多头压缩功能时支持keyCache和valueCache最后一维不同。
valueCache	[num_blocks, block_size, num_head, v_head_size]	float16/ bf16	ND	当前layer的value cache，不开启多头压缩功能时支持keyCache和valueCache最后一维不同。
slotMapping	[num_tokens]	int32	ND	每个token key或value在cache中的存储偏移，即（ block_id * block_size + offset_in_block ），值域范围为(INT32_MIN, num_blocks * block_size)且不存在重复数值。取值为负数时不进行任何搬运操作。
kStrides	[3]	int64	ND	k tensor 步幅。kStrides最后一个元素等于1。默认NULL，跳转到k tensor的下一个元素的各个维度的步幅。维度数与intensor相同。
vStrides	[3]	int64	ND	v tensor 步幅。vStrides最后一个元素等于1。默认NULL，跳转到v tensor的下一个元素的各个维度的步幅。维度数与intensor相同。
kOffset	[1]	int64	ND	默认0，k tensor相对于内存首地址的偏移。
vOffset	[1]	int64	ND	默认0，v tensor相对于内存首地址的偏移。

输出

参数	维度	数据类型	格式	描述
keyCacheOut	[num_blocks, block_size, num_head, k_head_size]	float16/ bf16	ND	所有的key cache。与输入keyCache为同一地址。
valueCacheOut	[num_blocks, block_size, num_head, v_head_size]	float16/ bf16	ND	所有的value cache。与输入valueCache为同一地址。

 版权所有 © 华为技术有限公司

10.54.5.2 开启单入单出功能

kvCacheCfg=K_CACHE_V_BYPASS，compressType=COMPRESS_TYPE_UNDEFINED

输入

参数	维度	数据类型	格式	描述
key	[num_tokens, num_head, k_head_size]	float16/ bf16	ND	当前step多个token的key。
keyCache	[num_blocks, block_size, num_head, k_head_size]	float16/ bf16	ND	所有的key cache。
slotMapping	[num_tokens]	int32	ND	• 每个token key或value在cache中的存储偏移，即（block_id * block_size + offset_in_block） • 取值范围为[0, num_blocks * kv_heads * block_size)，且不能存在重复值。即意味着num_tokens <= num_blocks*block_size。
kStrides	[3]	int64	ND	k tensor 步幅。kStrides最后一个元素等于1。默认NULL，跳转到k tensor的下一个元素的各个维度的步幅。维度数与输入tensor相同。
kOffset	[1]	int64	ND	默认0，k tensor相对于内存首地址的偏移。

输出

参数	维度	数据类型	格式	描述
keyCacheOut	[num_blocks, block_size, k_num_head, k_head_size]	float16/ bf16	ND	所有的key cache。与输入keyCache为同一地址。

10.54.6 规格约束

- ntokens = sum(seqlen[i])，i=0、1...batch-1。

 版权所有 © 华为技术有限公司

- 在key和value最后一维不同的情况下只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品，此场景下keyCache, valueCache, keyCacheOut, valueCacheOut只支持ND格式。
- key、value、keyCache、valueCache、keyCacheOut 、valueCacheOut的数据类型都必须完全一致。
- keyCache、valueCache、和keyCacheOut 、valueCacheOut的数据格式必须一致。
- 单入单出功能只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品，且不支持压缩场景。
- 压缩场景下cache的大小要大于压缩后的大小。
- 不支持Atlas 推理系列产品。
- 当前仅支持key、value在后两维连续的情况。

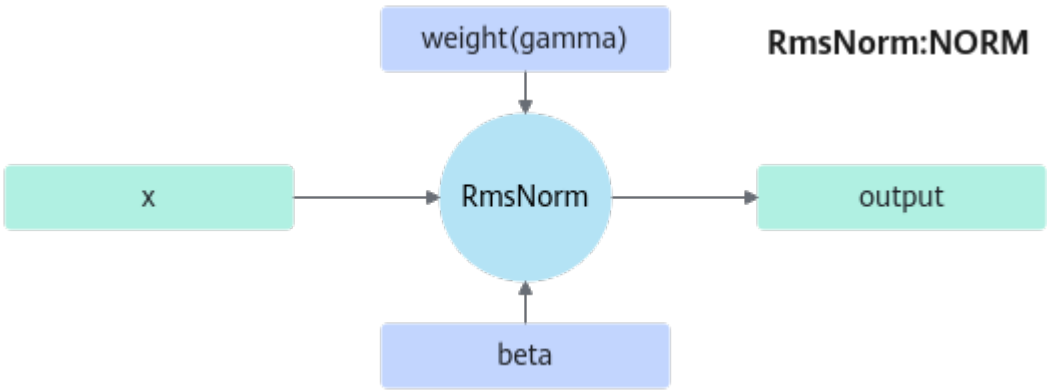
10.55 RmsNormOperation

10.55.1 功能描述

RMSNorm是一种归一化方法，它通过对数据的root mean square（RMS）进行归一化，避免了均值的使用。RMSNorm通过对数据的均值和方差进行调整，旨在优化深度学习模型的性能。RMSNorm不仅可以简化计算，还可以提高模型的收敛速度和稳定性。

当前支持Norm，PreNorm，PostNorm三种归一化操作。

图 10-54 NORM

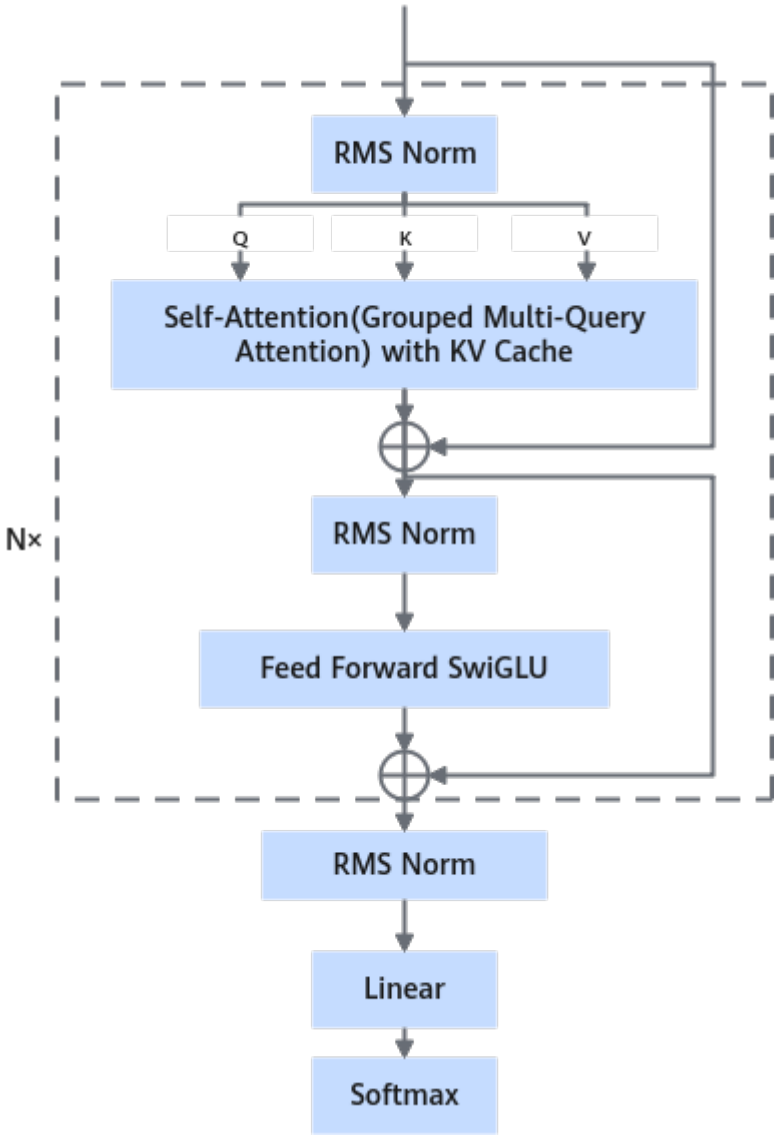


10.55.2 使用场景

通常使用在注意力机制attention前和attention与前馈网络ffn之间。

 版权所有 © 华为技术有限公司

图 10-55 RmsNormOperation 算子上下文



10.55.3 计算公式

RmsNorm

$$y = \frac{x}{\sqrt{\frac{1}{N}\sum_{1}^{N} x_i^2 + epsilon}} * gamma + beta$$

beta在rmsnorm quant量化场景下会使用。

 版权所有 © 华为技术有限公司

add+RmsNorm(PreNorm, PostNorm)

$$x^{'} = x + residual$$

$$y = \frac{x^{'}}{\sqrt{\frac{1}{N}\sum_{1}^{N} x_i^2 + epsilon}} * gamma + beta$$

其中gamma为可训练参数，epsilon是一个取值范围为(0, 1]的浮点值，用于避免除零。

量化

$$outputQuant = \frac{rmsNormOutput}{scale} + offset$$

10.55.4 定义

```
struct RmsNormParam {
    enum RmsNormType : int {
        RMS_NORM_UNDEFINED = 0,
        RMS_NORM_NORM,
        RMS_NORM_PRENORM,
        RMS_NORM_POSTNORM,
    };
    enum PrecisionMode : int {
        HIGH_PRECISION_MODE = 0,
        HIGH_PERFORMANCE_MODE,
    }
    enum ModelType : int {
        LLAMA_MODEL = 0,
        GEMMA_MODEL,
    };
    struct NormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        double layerNormEps = 1e-5;
        bool rstd = false;
        PrecisionMode precisionMode = HIGH_PRECISION_MODE;
        ModelType modelType = LLAMA_MODEL;
        DynamicQuantType dynamicQuantType = DYNAMIC_QUANT_UNDEFINED;
        uint8_t rsv[32] = {0};
    };
    struct PreNormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        bool hasBias = false;
        uint8_t rsv[23] = {0};
    };
    struct PostNormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        bool hasBias = false;
        uint8_t rsv[23] = {0};
    };
```

 版权所有 © 华为技术有限公司

```
    RmsNormType layerType = RMS_NORM_UNDEFINED;
    NormParam normParam;
    PreNormParam preNormParam;
    PostNormParam postNormParam;
    uint8_t rsv[8] = {0};
};
```

10.55.5 参数列表

成员名称	描述
RmsNormParam	RMSNORM参数。 • RMS_NORM_UNDEFINED：默认值，未定义。此默认类型不可用，用户需配置此项参数。 • RMS_NORM_NORM：NORM参数。 • RMS_NORM_PRENORM ： PRENORM参数。 • RMS_NORM_POSTNORM：POSTNORM参数。
PrecisionMode	PRECISIONMODE参数。 • HIGH_PRECISION_MODE：默认值，中间计算使用float类型。 • HIGH_PERFORMANCE_MODE：中间计算使用float16类型。
ModelType	MODELTYPE参数。 • LLAMA_MODEL：默认值，使用Llama rmsnorm的公式。 • GEMMA_MODEL：使用Gemma rmsnorm的公式。
layerType	归一化类型，参数如下： • RMS_NORM_UNDEFINED：默认值，未定义。 • RMS_NORM_NORM：NORM参数。 • RMS_NORM_PRENORM：PRENORM参数。 • RMS_NORM_POSTNORM：POSTNORM参数。
normParam	NORM参数。具体请参见**表10-8**。
preNormParam	PRENORM参数。具体请参见**表10-9**。
postNormParam	POSTNORM参数。具体请参见**表10-10**。
rsv[8]	预留参数。

 版权所有 © 华为技术有限公司

表 10-8 NormParam 成员

参数	类型	默认值	描述
quantType	QuantType	QUANT_UNQUANT	量化类型。 当前支持以下类型。 • QUANT_UNQUANT • QUANT_INT8
epsilon	float	1e-5	Epsilon，归一化时加在分母上防止除零。
layerNormEps	float	1e-5	Epsilon，默认为1e-5，暂时不使用。
rstd	bool	false	默认为False，设置为true时会使用训练的rmsnormforward算子。 仅Atlas 800I A2 推理产品/Atlas A2 训练系列产品支持该算子。 不支持和“precisionMode”，“modelType”同时设置。 量化场景下不支持使用“rstd”。
precisionMode	PrecisionMode	HIGH_PRECISION_MODE	默认为HIGH_PRECISION_MODE。 支持参数如下： • HIGH_PRECISION_MODE：默认值，中间计算使用float类型。 • HIGH_PERFORMANCE_MODE： 中间计算使用float16类型。 不支持和“rstd”，“modelType”同时设置。 输入类型只支持float16。 量化场景下不支持使用“precisionMode”，该场景下配置该参数将返回报错ERROR_INVALID_PARAM。
modelType	ModelType	LLAMA_MODEL	默认为LLAMA_MODEL，设置为GEMMA_MODEL时使用gemma模型的rmsnorm计算公式。 支持参数如下： • LLAMA_MODEL：默认值， Llama的rms norm计算公式。 • GEMMA_MODEL：Gemma的rms norm计算公式。 不支持和“rstd”，“precisionMode”同时启用。 量化场景下不支持使用“modelType”，该场景下配置该参数将返回报错ERROR_INVALID_PARAM。

 版权所有 © 华为技术有限公司

参数	类型	默认值	描述
dynamicQuantType	DynamicQuantType	DYNAMIC_QUANT_UNDEFINED	动态量化类型。默认为DYNAMIC_QUANT_UNDEFINED非动态量化。当前版本暂不支持非对称动态量化。
rsv[32]	uint8_t	{0}	预留参数。

说明

8.0.RC2及后续版本normParam不再支持quantInputScale、quantInputOffset量化参数。

表 10-9 PreNormParam 成员

参数	类型	默认值	描述
quantType	QuantType	QUANT_UNQUANT	量化类型。 当前支持以下类型。 • QUANT_UNQUANT • QUANT_INT8
epsilon	float	1e-5	Epsilon，归一化时加在分母上防止除零。
hasBias	bool	false	是否叠加偏置。默认为False，当需要输入beta时设置为True。 量化场景下不支持使用“hasBias”，该场景下配置该参数将返回报错ERROR_INVALID_PARAM。
rsv[23]	uint8_t	{0}	预留参数。

说明

8.0.RC2及后续版本PreNormParam不再支持quantInputScale、quantInputOffset量化参数。

表 10-10 PostNormParam 成员

参数	类型	默认值	描述
quantType	QuantType	QUANT_UNQUANT	量化类型。 当前支持以下类型： • QUANT_UNQUANT。 • QUANT_INT8。
epsilon	float	1e-5	Epsilon，归一化时加在分母上防止除零。

 版权所有 © 华为技术有限公司

参数	类型	默认值	描述
hasBias	bool	false	是否叠加偏置。默认为False，当需要输入beta时设置为True。 量化场景下不支持使用“hasBias”，该场景下配置该参数将返回报错ERROR_INVALID_PARAM。
rsv[23]	uint8_t	{0}	预留参数。

10.55.6 输入输出

RMS_NORM_NORM

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/ float/bf16	ND	float数据类型仅在rstd为true时支持，且rstd为true时只能使用float数据类型。 所有Tensor最后一维的dim_n大小需保持一致，其他维度的大小没有约束。 最后一维的大小需进行32字节对齐。
gamma	[1, ..., dim_n] / [dim_i, ..., dim_n] (0 <= i <= n)	float16/ float/bf16	ND	最后一维的大小需进行32字节对齐。 normParam.rstd为true时，维度数需要大于0，并小于x的维度数，gamma的维度从最后一维向前，每一维都需要和x保持一致。 rstd为false时，维度为[dim_n]。 数据类型与x保持一致。 dim_n前可以有任意个1。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
beta	[1, ..., dim_n]	float16/ float/bf16	ND	quantType = INT8时输入。 最后一维的大小要32字节对齐。 数据类型与x保持一致。 dim_n前可以有任意个1。 维度需要和gamma保持一致。
scale	[1]	float16/ bf16	ND	当quantType为QUANT_INT8时才输入。数据类型与x保持一致。
offset	[1]	int8	ND	当quantType为QUANT_INT8时才输入。
output	[dim_0, dim_1，... ，dim_n]	float16/ float/int8/ bf16	ND	quantType=INT8时输出int8。
rstd	[dim_0, ... ,1]	float16/ float/bf16	ND	normParam.rstd为true时输出。第i维大于等于x的维度数减去gamma的维度数时，大小为1，否则和x中对应维度大小相等。
除scale、offset和rstd参数外，其余tensor最后一维的n大小需保持一致。				

RMS_NORM_NORM DYNAMIC_QUANT

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16	ND	最后一维n的大小要32字节对齐，且小于等于12288。
gamma	[dim_n]	float16	ND	最后一维n的大小要32字节对齐，且小于等于12288。
beta	[dim_n]	float16	ND	最后一维n的大小要32字节对齐，且小于等于12288。
output	[dim_0, dim_1, ..., dim_n]	int8	ND	最后一维n的大小要32字节对齐，且小于等于12288。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
scale	[dim_0, dim_1, ..., dim_n-1]	float	ND	当quantType == QUANT_INT8且dynamicQuantType != DYNAMIC_QUANT_UNDEFINED时输出。
offset	[dim_0, dim_1, ..., dim_n-1]	float	ND	当quantType == QUANT_INT8且dynamicQuantType == DYNAMIC_QUANT_ASYMMETRIC时输出。 当前版本暂不支持。

RMS_NORM_PRENORM quantType = QUANT_UNQUANT

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	最后一维的大小要32字节对齐。
beta	[1, ..., dim_n]	float16/bf16	ND	当hasBias为True时输入。 最后一维的大小要32字节对齐。 dim_n前可以有任意个1。
residual	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	最后一维的大小要32字节对齐。
gamma	[1, ..., dim_n]	float16/bf16	ND	最后一维的大小要32字节对齐。 dim_n前可以有任意个1。 维度需要与beta保持一致。
output	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输出tensor，维度数和各维度值与x一致。
resOut	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	输出tensor，维度数和各维度值与x一致。
所有tensor最后一维的n大小需保持一致。				

 版权所有 © 华为技术有限公司

RMS_NORM_PRENORM quantType = QUANT_INT8

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16	ND	最后一维的大小要32字节对齐。
residual	[dim_0, dim_1, ..., dim_n]	float16	ND	最后一维的大小要32字节对齐。
gamma	[1, ..., dim_n]	float16	ND	最后一维的大小要32字节对齐。 dim_n前可以有任意个1。
beta	[1, ..., dim_n]	float16	ND	最后一维的大小要32字节对齐。 dim_n前可以有任意个1。 维度需要与gamma保持一致。
scale	[1]	float16	ND	量化scale参数。
offset	[1]	int8	ND	量化offset参数。
outputQuant	[dim_0, dim_1, ..., dim_n]	int8	ND	输出tensor，维度数和各维度值与x一致。
output	[dim_0, dim_1, ..., dim_n]	float16	ND	输出tensor，维度数和各维度值与x一致。
除scale和offset参数外，其余tensor最后一维的n大小需保持一致。				

RMS_NORM_POSTNORM quantType = QUANT_UNQUANT

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	当前支持quantType = QUANT_UNQUANT。 最后一维的大小要32字节对齐。
beta	[1, ..., dim_n]	float16/bf16	ND	当hasBias为True时输入。 最后一维的大小要32字节对齐。 dim_n前可以有任意个1。
residual	[dim_0, dim_1, ..., dim_n]	float16/bf16	ND	最后一维的大小要32字节对齐。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
gamma	[1, ..., dim_n]	float16/ bf16	ND	最后一维的大小要32字节对齐。 dim_n前可以有任意个1。 维度需要与beta保持一致。
output	[dim_0, dim_1, ..., dim_n]	float16/ bf16	ND	输出tensor，维度数和各维度值与x一致。
所有tensor最后一维的n大小需保持一致。				

RMS_NORM_POSTNORM quantType = QUANT_INT8

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/ bf16	ND	输入tensor0，最后一维的大小要32字节对齐。所有Tensor最后一维的n大小需保持一致。
residual	[dim_0, dim_1, ..., dim_n]	float16/ bf16	ND	输入tensor1，最后一维的大小要32字节对齐。
gamma	[dim_n]/[1, dim_n]	float16/ bf16	ND	输入tensor2，最后一维的大小要32字节对齐。
scale	[1]	float16/ bf16	ND	输入tensor3，量化scale参数。
offset	[1]	int8	ND	输入tensor4，量化offset参数。
outputQuant	[dim_0, dim_1, ..., dim_n]	int8	ND	输出tensor0，维度数和各维度值与x一致。最后一维的大小要32字节对齐。
output	[dim_0, dim_1, ..., dim_n]	float16/ bf16	ND	输出tensor1，维度数和各维度值与x一致。最后一维的大小要32字节对齐。
除scale和offset参数外，其余tensor最后一维的n大小需保持一致，dim_n=32*n（n≥1）。				

10.55.7 规格约束

- 除scale，offset，rstd tensors外，所有输入输出tensor的最后一维大小相等，需要32字节对齐。
- rstd仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品，不能和precisionMode，modelType同时设置，不支持量化场景使用。

 版权所有 © 华为技术有限公司

- precisionMode不能和rstd，modelType同时设置，不支持量化场景使用。
- modelType不能和rstd，precisionMode同时设置，不支持量化场景使用。
- dynamicQuantType目前只支持对称动态量化DYNAMIC_QUANT_SYMMETRIC。动态对称量化时，输入的最后一维大小需要小于等于12288。
- Atlas 训练系列产品目前仅支持非量化非融合rmsnorm。
- RMS_NORM_PRENORM、RMS_NORM_POSTNORM量化场景仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- Atlas 200I/500 A2 推理产品上仅支持float16数据类型，且layerType为RMS_NORM_NORM、quantType为QUANT_UNQUANT、QUANT_INT8，dynamicQuantType为0，即不支持dynamicQuant。

10.55.8 功能列表

RmsNorm

- **非量化**
  - 训练rmsnormforward算子，设置rstd为true开启。
  - precisionMode默认HIGH_PRECISION_MODE，中间计算使用float类型。HIGH_PERFORMANCE_MODE，在最后做gamma（weight）的乘法计算的时候使用float16类型。
  - modelType默认LLAMA_MODEL，使用llama模型的rmsnorm公式。GEMMA_MODEL，使用gemma模型的rmsnorm公式。区别在乘gamma（weight）前，会将gamma+1再做乘法。
- **量化**
  - rmsnorm量化。
  - dynamicQuantType默认值DYNAMIC_QUANT_UNDEFINED（普通量化）DYNAMIC_QUANT_SYMMETRIC 动态对称量化。

add+RmsNorm 融合（Prenorm，PostNorm）在做 rmsnorm 前，将 x 和 residual 相加

- **非量化**
  - PreNorm返回add+residual的结果，PostNorm不返回
  - hasBias参数。为true时，可以输入bias tensor。为false时，不输入bias tensor。
- **量化**

  只有Prenorm支持。

10.55.9 算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
```

 版权所有 © 华为技术有限公司

```
#include "atb/atb_infer.h"

#include "demo_util.h"

const int32_t DEVICE_ID = 0;
const uint32_t DIM_0 = 4;
const uint32_t DIM_1 = 1024;
const uint32_t DIM_2 = 5120;

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建shape为[4, 1024, 5120]的tensor
    atb::Tensor x = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>(DIM_0 * DIM_1 * DIM_2, 2.0),
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {DIM_0, DIM_1, DIM_2});
    atb::Tensor gamma = CreateTensorFromVector(
        contextPtr, stream, std::vector<float>(DIM_2, 2.0), ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{DIM_2});
    atb::SVector<atb::Tensor> inTensors = {x, gamma};
    return inTensors;
}

/**
 * @brief 创建一个非量化rmsnorm operation
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *CreateRmsNormOperation()
{
    atb::infer::RmsNormParam param;
    param.layerType = atb::infer::RmsNormParam::RmsNormType::RMS_NORM_NORM;
    param.normParam.quantType = atb::infer::QuantType::QUANT_UNQUANT;
    atb::Operation *rmsNormOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &rmsNormOp));
    return rmsNormOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    atb::Context *context = nullptr;
    void *stream = nullptr;

    CHECK_STATUS(aclInit(nullptr));
    CHECK_STATUS(aclrtSetDevice(DEVICE_ID));
    CHECK_STATUS(atb::CreateContext(&context));
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 创建op
    atb::Operation *rmsnormOp = CreateRmsNormOperation();
    // 准备输入tensor
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {DIM_0, DIM_1,
DIM_2});
    variantPack.outTensors = {tensorOut};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspace大小
```

 版权所有 © 华为技术有限公司

```
    CHECK_STATUS(rmsnormOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // rmsnorm执行
    rmsnormOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放资源
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    CHECK_STATUS(aclrtFree(tensorOut.deviceData));
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(rmsnormOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    std::cout << "Rmsnorm demo success!" << std::endl;
    return 0;
}
```

10.56 RmsNormWithStrideOperation

功能

为了使RmsNorm支持非连续Tensor，将x tensor通过（shape、strides、offset）表示。strides描述某个维度上相邻两个元素的间隔，offset表示这个Tensor的首元素相对addr的偏移。

示例：

例如现有一个shape=(6, 5)、strides=(10, 1)、offset=22的Tensor，其内存排布如下：

 版权所有 © 华为技术有限公司

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$	$a_{0,8}$	$a_{0,9}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$	$a_{1,8}$	$a_{1,9}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$	$a_{2,8}$	$a_{2,9}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$	$a_{3,8}$	$a_{3,9}$
$a_{4,0}$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$a_{4,7}$	$a_{4,8}$	$a_{4,9}$
$a_{5,0}$	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	$a_{5,7}$	$a_{5,8}$	$a_{5,9}$
$a_{6,0}$	$a_{6,1}$	$a_{6,2}$	$a_{6,3}$	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	$a_{6,7}$	$a_{6,8}$	$a_{6,9}$
$a_{7,0}$	$a_{7,1}$	$a_{7,2}$	$a_{7,3}$	$a_{7,4}$	$a_{7,5}$	$a_{7,6}$	$a_{7,7}$	$a_{7,8}$	$a_{7,9}$
$a_{8,0}$	$a_{8,1}$	$a_{8,2}$	$a_{8,3}$	$a_{8,4}$	$a_{8,5}$	$a_{8,6}$	$a_{8,7}$	$a_{8,8}$	$a_{8,9}$
$a_{9,0}$	$a_{9,1}$	$a_{9,2}$	$a_{9,3}$	$a_{9,4}$	$a_{9,5}$	$a_{9,6}$	$a_{9,7}$	$a_{9,8}$	$a_{9,9}$

即该Tensor排布如上图的深色位置。这个完整的Tensor在内存排布上是不连续的，strides描述Tensor维度上相邻两个元素的间隔，如果在维度1上的stride为1， 该维度是连续的；如果在维度0上的stride为10，那么相邻的元素间隔10个元素，即非连续。offset表示这个Tensor的首元素相对addr的偏移。

计算公式

$$y = \frac{x}{\sqrt{\frac{1}{N}\sum_{1}^{N} x_i^2 + epsilon}} * gamma$$

其中gamma为可训练参数，epsilon是一个取值范围为(0, 1]的浮点值，用于避免除零。

定义

```
struct RmsNormWithStrideParam {
    enum RmsNormType : int {
        RMS_NORM_UNDEFINED = 0,
        RMS_NORM_NORM,
        RMS_NORM_PRENORM,
        RMS_NORM_POSTNORM,
    };
    enum PrecisionMode : int {
        HIGH_PRECISION_MODE = 0,
```

 版权所有 © 华为技术有限公司

```
        HIGH_PERFORMANCE_MODE,
    };
    enum ModelType : int {
        LLAMA_MODEL = 0,
        GEMMA_MODEL,
    };
    struct NormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        double layerNormEps = 1e-5;
        bool rstd = false;
        PrecisionMode precisionMode = HIGH_PRECISION_MODE;
        ModelType modelType = LLAMA_MODEL;
        DynamicQuantType dynamicQuantType = DYNAMIC_QUANT_UNDEFINED;
        uint8_t rsv[32] = {0};
    };
    struct PreNormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        bool hasBias = false;
        uint8_t rsv[23] = {0};
    };
    struct PostNormParam {
        QuantType quantType = QUANT_UNQUANT;
        float epsilon = 1e-5;
        bool hasBias = false;
        uint8_t rsv[23] = {0};
    };
    RmsNormType layerType = RMS_NORM_UNDEFINED;
    NormParam normParam;
    PreNormParam preNormParam;
    PostNormParam postNormParam;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	描述
layerType	归一化类型。 • RMS_NORM_UNDEFINED：默认值，未定义。 • RMS_NORM_NORM：NORM参数。 • RMS_NORM_PRENORM：PRENORM参数。 • RMS_NORM_POSTNORM：POSTNORM参数。
normParam	NORM参数。具体请参见**表10-11**。
preNormParam	PRENORM参数。具体请参见**表10-12**。
postNormParam	POSTNORM参数。具体请参见**表10-13**。
rsv[8]	预留参数。

 版权所有 © 华为技术有限公司

表 10-11 normParam 成员

成员名称	类型	默认值	取值范围	是否必选	描述
quantType	QuantType	QUANT_UNQUANT	QUANT_UNQUANT	是	在算子的计算完成后进行量化。 当前仅支持以下类型：QUANT_UNQUANT。
epsilon	float	1e-5	1e-5	是	Epsilon，归一化时加在分母上防止除0。
layerNormEps	double	1e-5	1e-5	是	Epsilon，默认为1e-5，暂不使用。
rstd	bool	false	true/false	是	默认为False，不支持和“precisionMode”，“modelType”同时设置。量化场景下不支持使用“rstd”。
precisionMode	PrecisionMode	HIGH_PRECISION_MODE	HIGH_PRECISION_MODE HIGH_PERFORMANCE_MODE	是	• HIGH_PRECISION_MODE：默认值，中间计算使用float类型。 • HIGH_PERFORMANCE_MODE： 中间计算使用float16类型。 不支持和“rstd”，“modelType”同时设置。输入类型只支持float16。 量化场景下不支持使用“precisionMode”，该场景下配置该参数将返回报错ERROR_INVALID_PARAM。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	是否必选	描述
modelType	ModelType	LLAMA_MODEL	LLAMA_MODEL GEMMA_MODEL	是	设置为GEMMA_MODEL时使用gemma模型的rmsnorm计算公式。 ● LLAMA_MODEL：默认值， Llama的rms norm计算公式。 ● GEMMA_MODEL：Gemma的rms norm计算公式。 不支持和“rstd”，“precisionMode”同时启用。 量化场景下不支持使用“modelType”，该场景下配置该参数将返回报错ERROR_INVALID_PARAM。
dynamicQuantType	DynamicQuantType	DYNAMIC_QUANT_UNDEFINED	DYNAMIC_QUANT_UNDEFINED	否	只支持DYNAMIC_QUANT_UNDEFINED非动态量化。
rsv[32]	uint8_t	{0}	[0]	否	预留参数。

表 10-12 PreNormParam 成员

成员名称	类型	默认值	取值范围	是否必选	描述
quantType	QuantType	QUANT_UNQUANT	QUANT_UNQUANT QUANT_INT8	是	量化类型。
epsilon	float	1e-5	1e-5	是	Epsilon，归一化时加在分母上防止除零。
hasBias	bool	false	true/false	否	是否叠加偏置。当需要输入beta时设置为True。量化场景下不支持使用“hasBias”，该场景下配置该参数将返回报错ERROR_INVALID_PARAM。
rsv[23]	uint8_t	{0}	[0]	否	预留参数。

 版权所有 © 华为技术有限公司

表 10-13 PostNormParam 成员

成员名称	类型	默认值	取值范围	是否必选	描述
quantType	QuantType	QUANT_UNQUANT	QUANT_UNQUANT	是	量化类型。当前仅支持QUANT_UNQUANT。
epsilon	float	1e-5	1e-5	是	Epsilon，归一化时加在分母上防止除零。
hasBias	bool	false	true/false	否	是否叠加偏置。当需要输入beta时设置为True。
rsv[23]	uint8_t	{0}	[0]	否	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0, ..., dim_n-1, dim_n]	float16/bf16	ND	最后一位的大小需要为16的倍数。
gamma	[1, dim_n]/[dim_i, ..., dim_n] (0<=i<=n)	float16/bf16	ND	维度为[1, dim_n]，最后一维和x保持一致。
xStrides	[x_shape]	int64	ND	x的stride步长，x_shape表示x的维度数，大于0，传入方式为host tensor，最后一个数需为1。
xOffset	[1]	int64	ND	x的offset，需大于0，默认值为0，传入方式为host tensor。

输出

参数	维度	数据类型	格式	描述
output	[dim_0, ..., dim_n-1, dim_n]	float16/bf16	ND	维度和x保持一致。

规格约束

- 在NORM场景下不支持量化和rstd。

 版权所有 © 华为技术有限公司

- 只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 暂不支持preNormParam和postNormParam成员。

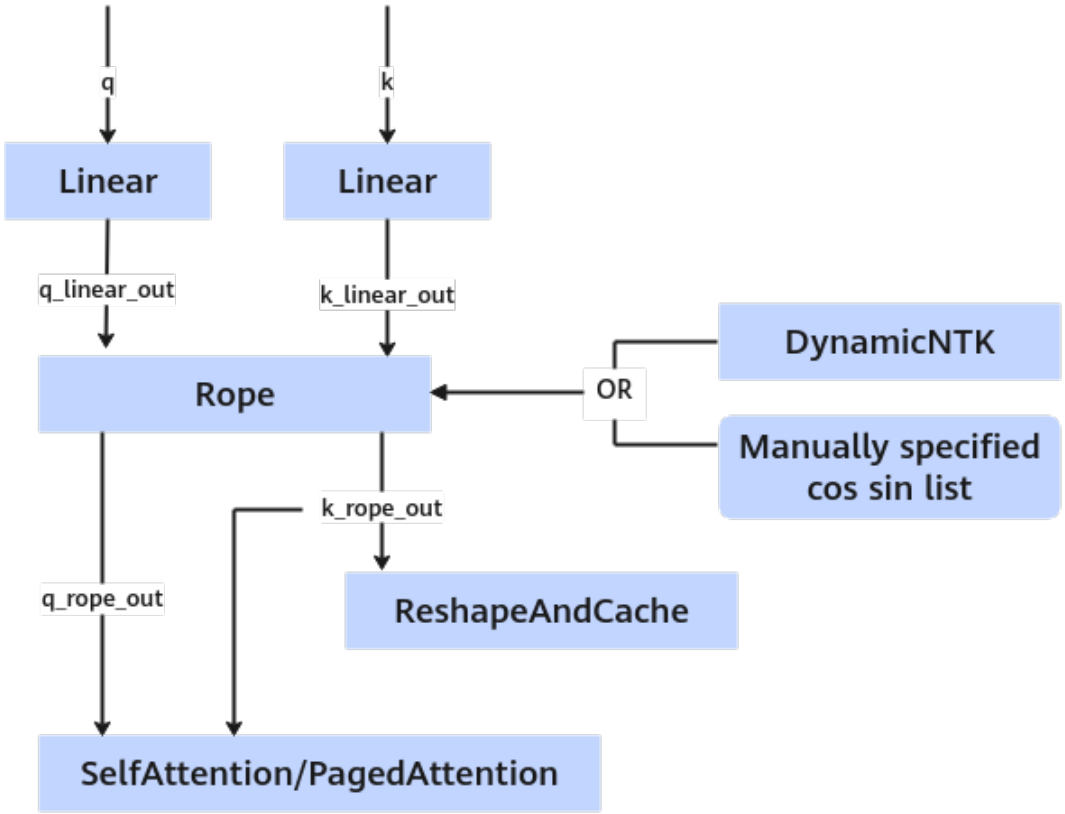
10.57 RopeOperation

功能

旋转位置编码（Rotary Position Embedding，RoPE），以旋转矩阵的方式在q、k中注入位置信息，使得attention计算时能感受到token的位置关系，在各大模型中，RoPE被广泛应用。RoPE以绝对位置编码的方式实现了相对位置编码，能有效保持位置信息相对关系，并且可以通过编码外推的方式支持超过训练长度的位置编码。

算子上下文

图 10-56 RopeOperation 算子上下文



计算公式

对于二维情况

$$f\left(q, m\right) = \begin{pmatrix} cos\, m\theta & -sin\, m\theta \\ sin\, m\theta & cos\, m\theta \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \end{pmatrix}$$

假设空间是偶数维的，把原始空间切分一个个正交的二维子空间，在上面做独立的不同角度的旋转，可以扩展到高维空间。

$$f\left(\begin{array}{c} q, m \end{array}\right) = \begin{pmatrix} cos\, m\theta_0 & -sin\, m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ sin\, m\theta_0 & cos\, m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & cos\, m\theta_1 & -sin\, m\theta_1 & \cdots & 0 & 0 \\ 0 & 0 & sin\, m\theta_1 & cos\, m\theta_1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & cos\, m\theta_{d/2-1} & -sin\, m\theta_{d/2-1} \\ 0 & 0 & 0 & 0 & \cdots & sin\, m\theta_{d/2-1} & cos\, m\theta_{d/2-1} \end{pmatrix}$$

定义

```
struct RopeParam {
    int32_t rotaryCoeff = 4;
    int32_t cosFormat = 0;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rotaryCoeff	int32_t	4	rope，旋转系数，对半旋转是2，支持配置2、4、head_size / 2、head_size。
cosFormat	int32_t	0	训练用参数，支持配置0或1。
rsv[8]	uint8_t	{0}	预留参数。

rotaryCoeff参数选择与原始计算公式的对应关系如下：

 版权所有 © 华为技术有限公司

$$f(q,m)=\begin{bmatrix} q_0 \\ q_1 \\ \dots \\ q_{d/2-1} \\ q_{d/2} \\ q_{d/2+1} \\ \dots \\ q_{d-1} \end{bmatrix} * \begin{bmatrix} cos(m\theta_0) \\ cos(m\theta_1) \\ \dots \\ cos(m\theta_{d/2-1}) \\ cos(m\theta_0) \\ cos(m\theta_1) \\ \dots \\ cos(m\theta_{d/2-1}) \end{bmatrix} + \begin{bmatrix} -q_{d/2} \\ -q_{d/2+1} \\ \dots \\ -q_{d-1} \\ q_0 \\ q_{d_i} \\ \dots \\ q_{d/2-1} \end{bmatrix} * \begin{bmatrix} sin(m \\ sin(m \\ \dots \\ sin(m\theta \\ sin(m \\ sin(m \\ \dots \\ sin(m\theta \end{bmatrix}$$

$$\theta_i = \frac{1}{10000^{\frac{2i}{d}}}$$

其中m为token的位置，d为query或key的维度。

- rotaryCoeff = 2的情况：

  如上式，query和key分别取一半，进行旋转操作后拼接，称为对半旋转，对应rotaryCoeff = 2，即1/2。

- rotaryCoeff = 4的情况：

  以此类推，rotaryCoeff = 4对应于1/4，表示query和key分成前后两半，每一半按rotaryCoeff = 2的情况处理。

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \dots \\ q_{d-2} \\ q_{d-1} \end{pmatrix} \otimes \begin{pmatrix} cosm\theta_0 \\ cosm\theta_0 \\ cosm\theta_1 \\ cosm\theta_1 \\ \dots \\ cosm\theta_{d/2-1} \\ cosm\theta_{d/2-1} \end{pmatrix} + \begin{pmatrix} -q_1 \\ q_0 \\ -q_3 \\ q_2 \\ \dots \\ -q_{d-1} \\ q_{d-2} \end{pmatrix} \otimes \begin{pmatrix} sinm\theta_0 \\ sinm\theta_0 \\ sinm\theta_1 \\ sinm\theta_1 \\ \dots \\ sinm\theta_{d/2-1} \\ sinm\theta_{d/2-1} \end{pmatrix}$$

- rotaryCoeff = head_size的情况：

 版权所有 © 华为技术有限公司

如上式，query和key中的维度两两一组交错排列，其中图示的cos和sin列表经过了concat操作，表现为连续相同的2个cos、sin值。

如果不经过concat操作，cos和sin列表的长度为上式中的一半，即head_size/2。算子也支持此种情况，此时需要将rotaryCoeff = head_size/2。

换言之，使用上图的计算模式只需要满足rotaryCoeff与cos或sin输入tensor的最后一维大小相等。

输入

参数	维度	数据类型	格式	描述
query	[ntokens, hiddenSizeQ]	float16/bf16	ND	当前step多个token的query。
key	[ntokens, hiddenSizeK]	float16/bf16	ND	当前step多个token的key。
cos	[ntokens, head_size] / [ntokens, head_size / 2]	float16/ float/bf16	ND	• 当cos的第二个维度与参数rotaryCoeff不相等时，其值为head_size。 • ROPE高精度模式，需要输入cos的数据类型为float时生效。
sin	[ntokens, head_size] / [ntokens, head_size/ 2]	float16/ float/bf16	ND	• 当sin的第二个维度与参数rotaryCoeff不相等时，其值为head_size。 • ROPE高精度模式，需要输入sin的数据类型为float时生效。
seqlen	[batch]	uint32/int32	ND	-

输出

参数	维度	数据类型	格式	描述
ropeQ	[ntokens, hiddenSizeQ]	float16/bf16	ND	旋转后的query。
ropeK	[ntokens, hiddenSizeK]	float16/bf16	ND	旋转后的key。

约束

- 输入tensor数据类型需保持一致，高精度模式例外。

 版权所有 © 华为技术有限公司

- cos、sin传入数据类型为float时，中间计算结果以float保存。
- hiddenSizeQ和hiddenSizeK必须是head_size的整数倍，满足hiddenSizeQ = head_size * headNumQ、hiddenSizeK = head_size * headNumK，其中headNumQ可以大于headNumK，hiddenSizeQ和hiddenSizeK需要32bytes对齐。
- ntokens = sum(seqlen[i])，i=0...batch-1。
- query和key要求两维，有部分模型使用了4维，这种情况下维度是：

  [batch, seqlen, headNum, head_size]；对应的ropeQ、ropeK也是四维，维度输入输出对应。
- Decoder阶段要取cos和sin表中seqlen对应的cos/sin值输入。
- 多batch场景需要组合使用gather算子。
- Atlas 200I/500 A2 推理产品上仅支持float16数据类型。

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"
#include "demo_util.h"
const uint32_t BATCH_SIZE = 1;    // 批处理大小
const uint32_t NTOKENS = 4;       // TOKEN大小
const uint32_t HIDDENSIZEQ = 16;  // Q 隐藏层大小
const uint32_t HIDDENSIZEK = 16;  // K 隐藏层大小
const uint32_t HEAD_SIZE = 8;     // 头大小
/**
 * @brief 创建一个ROPE的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::RopeParam opParam;
    atb::Operation *ropeOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(opParam, &ropeOp));
    return ropeOp;
}
/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    std::vector<float> qData(NTOKENS * HIDDENSIZEQ, 1.0);
    atb::Tensor tensorQ = CreateTensorFromVector(
        contextPtr, stream, qData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, HIDDENSIZEQ});
    std::vector<float> kData(NTOKENS * HIDDENSIZEK, 1.0);
    atb::Tensor tensorK = CreateTensorFromVector(
        contextPtr, stream, kData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, HIDDENSIZEK});
    std::vector<float> cos(NTOKENS * HEAD_SIZE, 1.0);
    atb::Tensor tensorCos =
        CreateTensorFromVector(contextPtr, stream, cos, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
```

 版权所有 © 华为技术有限公司

```
{NTOKENS, HEAD_SIZE});
    std::vector<float> sin(NTOKENS * HEAD_SIZE, 1.0);
    atb::Tensor tensorSin =
        CreateTensorFromVector(contextPtr, stream, sin, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
{NTOKENS, HEAD_SIZE});
    std::vector<int32_t> seqLenHost(BATCH_SIZE, 4);
    atb::Tensor tensorSeqLen = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    tensorSeqLen.hostData = seqLenHost.data();
    atb::SVector<atb::Tensor> inTensors = {tensorQ, tensorK, tensorCos, tensorSin, tensorSeqLen};
    return inTensors;
}
int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);
    // 算子实例
    atb::Operation *ropeOp = PrepareOperation();
    // 准备输入张量
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);
    // 准备输入张量ropeQ和ropeK
    atb::Tensor ropeQ = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS,
HIDDENSIZEQ});
    atb::Tensor ropeK = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS,
HIDDENSIZEK});
    variantPack.outTensors = {ropeQ, ropeK};
    uint64_t workspaceSize = 0;
    CHECK_STATUS(ropeOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // 算子执行
    ropeOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(ropeOp));
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));
    CHECK_STATUS(aclFinalize());
    std::cout << "Rope demo success!" << std::endl;
    return 0;
}
```

10.58 RopeQConcatOperation

功能

旋转位置编码后进行concat操作。

 版权所有 © 华为技术有限公司

计算公式

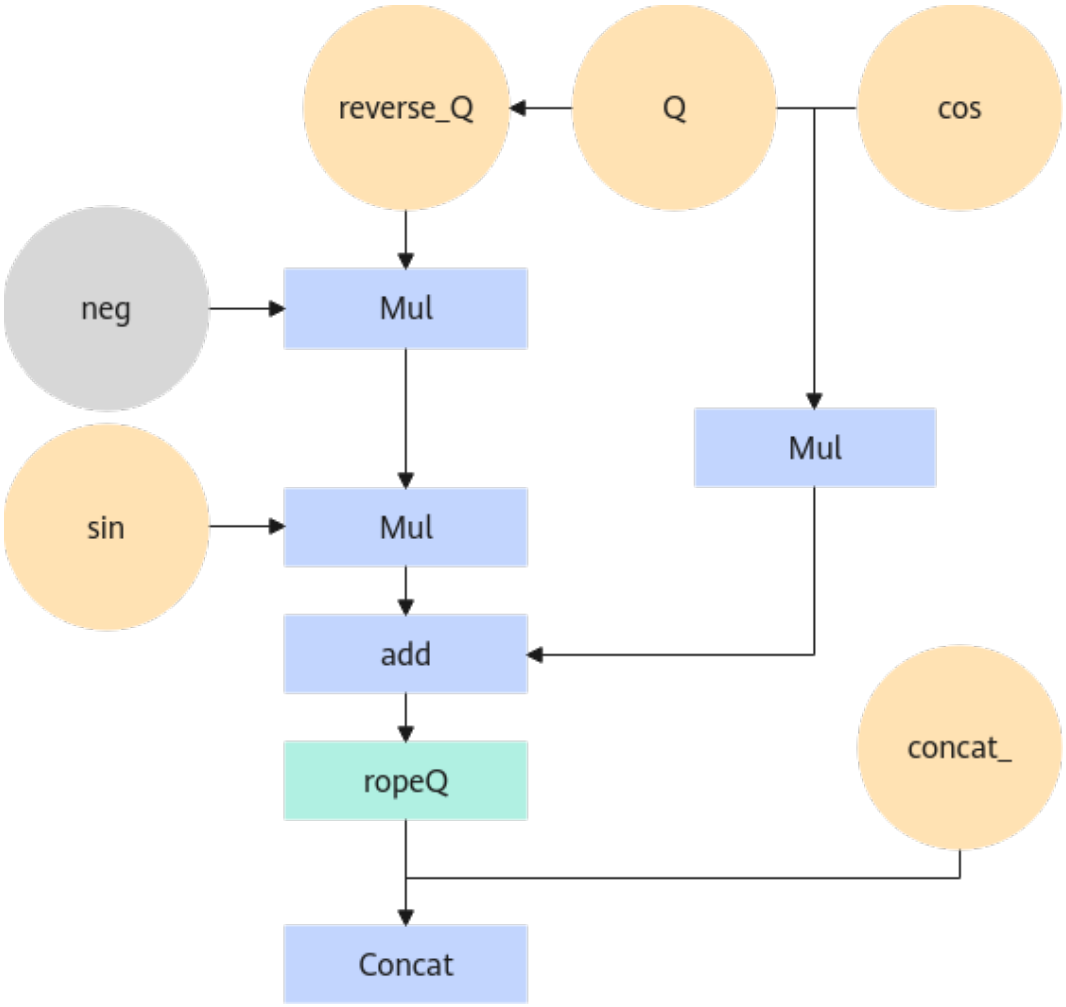
rope计算公式：

$$f(q,m)=\begin{bmatrix} q_0 \\ q_1 \\ \dots \\ q_{d/2-1} \\ q_{d/2} \\ q_{d/2+1} \\ \dots \\ q_{d-1} \end{bmatrix} * \begin{bmatrix} cos(m\theta_0) \\ cos(m\theta_1) \\ \dots \\ cos(m\theta_{d/2-1}) \end{bmatrix} + \begin{bmatrix} -q_{d/2} \\ -q_{d/2+1} \\ \dots \\ -q_{d-1} \end{bmatrix} * \begin{bmatrix} sin(m \\ sin(m \\ \dots \\ sin(m\theta \end{bmatrix} + \begin{bmatrix} cos(m\theta_0) \\ cos(m\theta_1) \\ \dots \\ cos(m\theta_{d/2-1}) \end{bmatrix} + \begin{bmatrix} q_0 \\ q_{d_1} \\ \dots \\ q_{d/2-1} \end{bmatrix} * \begin{bmatrix} sin(m \\ sin(m \\ \dots \\ sin(m\theta \end{bmatrix}$$

计算图

kernel计算流程如下图所示：

 版权所有 © 华为技术有限公司



定义

```
struct RopeQConcatParam {
    uint8_t rsv[16] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
rsv	uint8_t[]	{0}	[0]	否	预留字段。

输入

参数	维度	数据类型	格式	描述
q	[ntokens, hidden_size_q]	float16/ bf16	ND	2维。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
cos	[ntokens, head_dim]	与q一致	ND	2维。
sin	[ntokens, head_dim]	与q一致	ND	2维。
ConcatInput	[ntokens, head_num, concat_size]	与q一致	ND	3维。

输出

参数	维度	数据类型	格式	描述
ropeQConcat	[ntokens, head_num, head_dim+concat_size]	与q一致	ND	3维。

规格约束

- hidden_size_q = head_dim * head_num。
- head_dim*sizeof(dtype) 需要32Byte对齐，即head_dim需要是16的整数倍且需小于等于64。
- concat_size*sizeof(dtype) 需要32Byte对齐，即concat_size需要是16的整数倍。
- 保证headDim * 26 + concatSize * 2<maxUbSize(196352) ，对head_num无限制。
- 只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

10.59 SelfAttentionOperation

10.59.1 功能描述

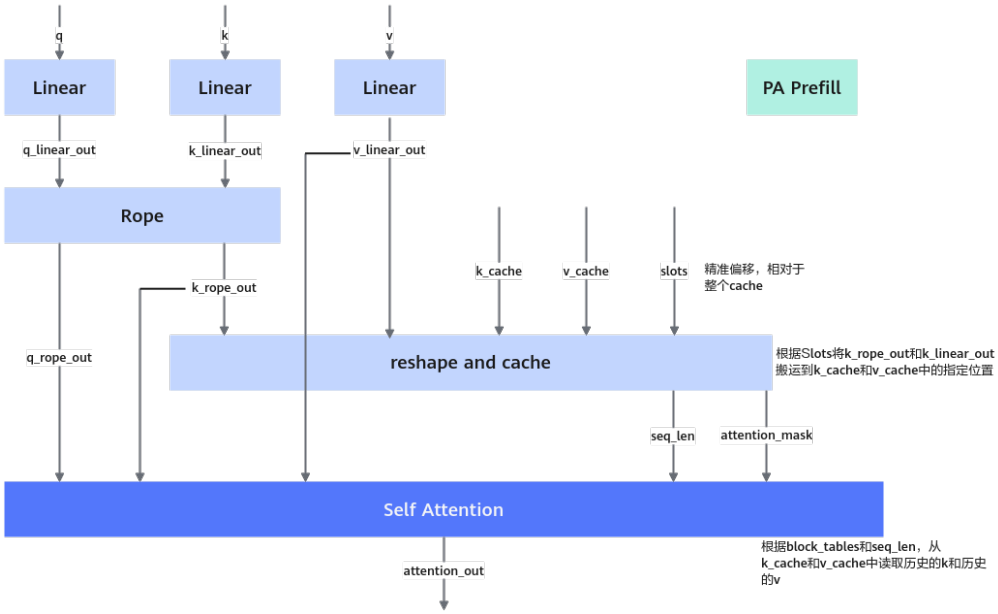
基于传统flash attention按照layerId管理kvcache的机制计算attention score。

10.59.2 算子上下文

- 当参数“calcType”置为PA_ENCODER时，为paged attention的全量阶段，如下图所示。

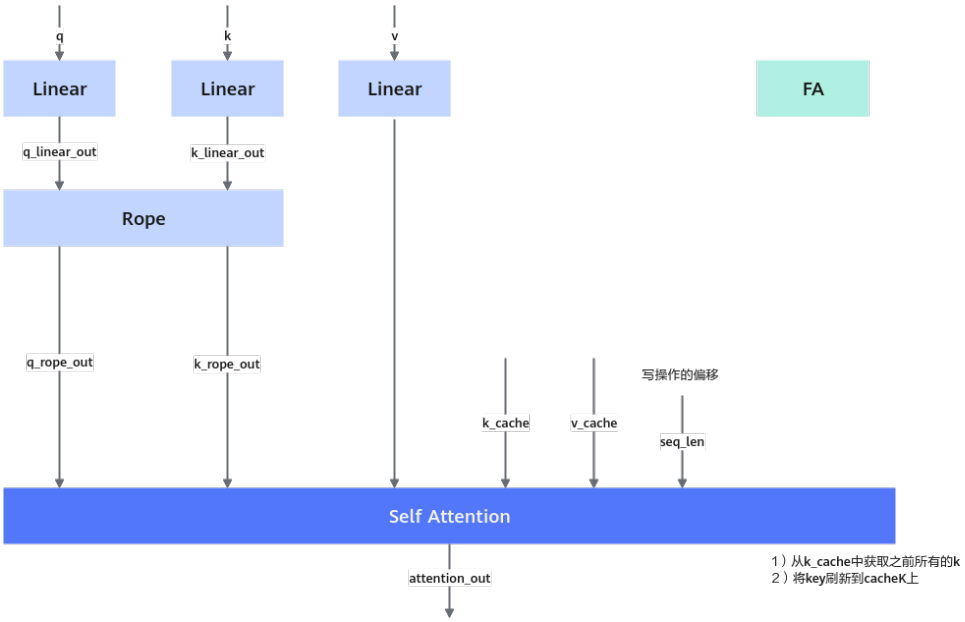
 版权所有 © 华为技术有限公司

图 10-57 paged attention 的全量阶段在模型中的位置



- 其他情况为传统flash attention的全量/增量阶段，如下图所示。

图 10-58 flash attention 在模型中的位置

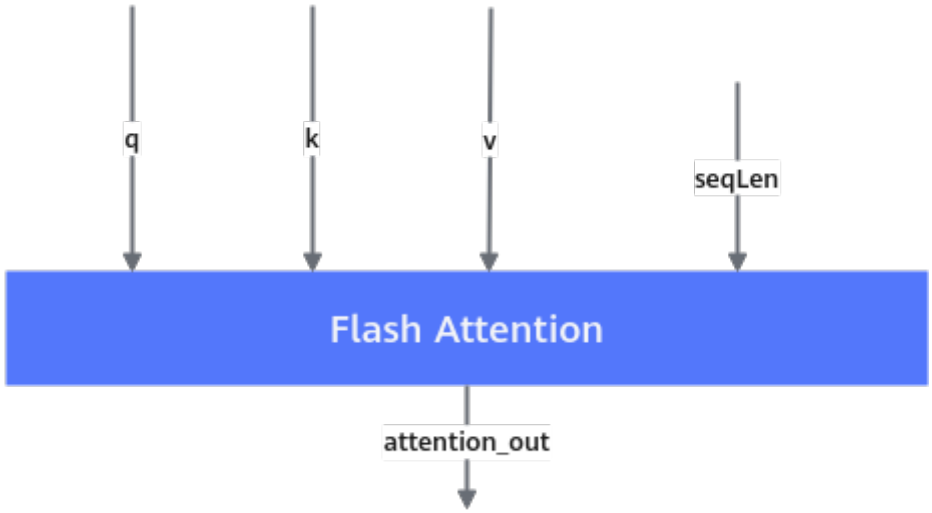


10.59.3 计算图

- 在不同参数配置下，加速库OP内部组图不同。其中，当参数“calcType”置为PA_ENCODER时，计算图如下图所示。

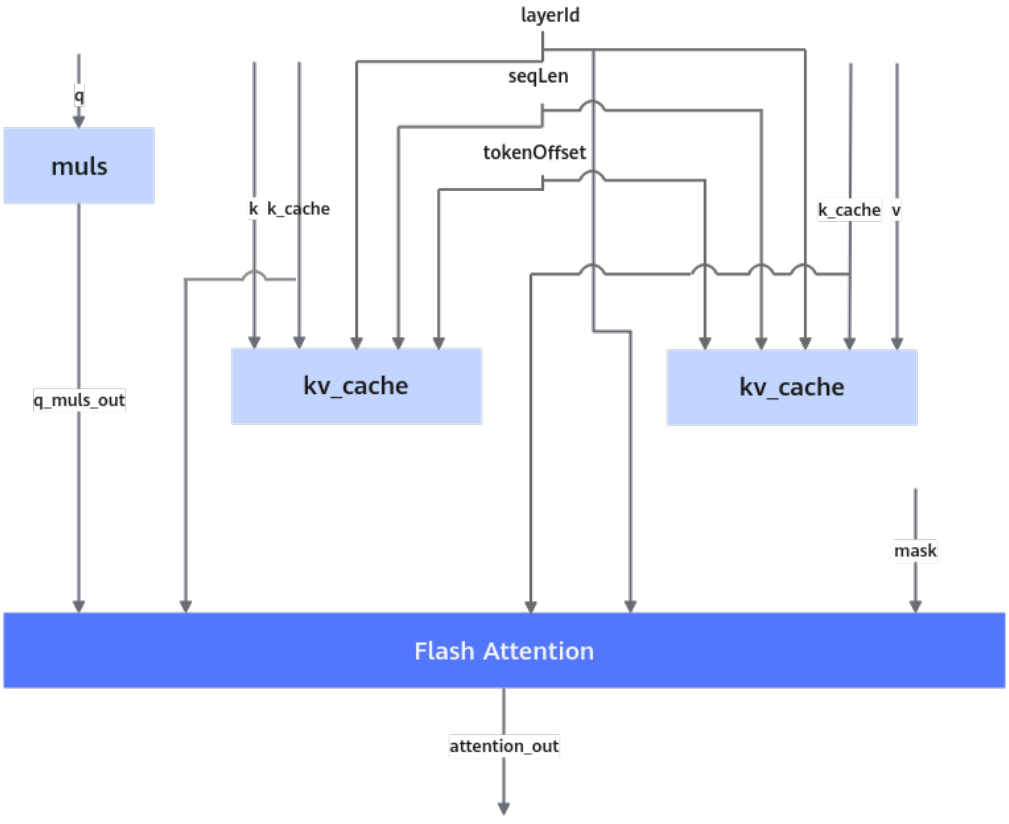
 版权所有 © 华为技术有限公司

图 10-59 paged attention 的全量阶段计算图



- 当参数calcType非PA_ENCODER，且参数“kvCacheCfg”置为K_CACHE_V_CACHE时，计算图如下图所示。

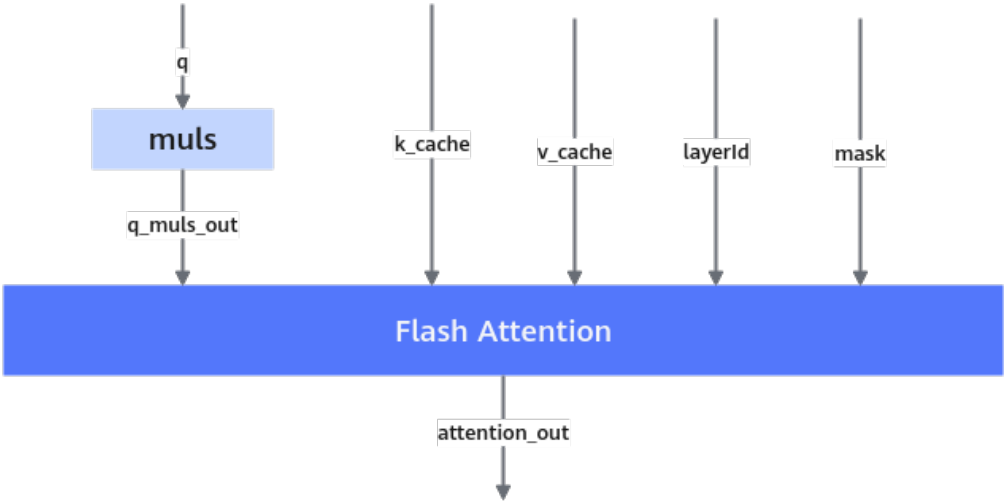
图 10-60 flash attentionkvCacheCfg 置为 K_CACHE_V_CACHE 时的计算图



- 当参数“calcType”非PA_ENCODER，且参数“kvCacheCfg”置为K_BYPASS_V_BYPASS时，计算图如下图所示。

 版权所有 © 华为技术有限公司

图 10-61 flash attentionkvCacheCfg 置为 K_BYPASS_V_BYPASS 时的计算图



10.59.4 计算公式

- $$Q = preprocess(input_q)$$

  preprocess包含对q的缩放，其缩放系数由参数qScale给出，以及当开启clamp缩放时，对q做torch.clamp。

- $$mask = preprocess(input_mask)$$

  当开启压缩mask特性时preprocess会对mask进行压缩。

- 当“calcType”非PA_ENCODER且“kvCacheCfg”不为K_BYPASS_V_BYPASS：

  $$K = kvcache(input_k, input_kcache)$$

  $$V = kvcache(input_v, input_vcache)$$

  这一步会将k，v刷新到kvcache中，具体操作可以见10.32 KVCacheOperation。

- $$attenOut = PostProcess(Softmax(Zoom(Q * K^T) + mask) * V)$$

  其中，Zoom函数包括了对 $Q*K^T$ 结果进行缩放，缩放系数由参数qkScale给出，一般为1 / sqrt(head_size)，当开启logN缩放时，缩放系数由传入的intensor logN 给出。PostProcess函数包含对结果的reshape。

10.59.5 定义

```
struct SelfAttentionParam {
    enum CalcType : int {
        UNDEFINED = 0,
        ENCODER,
        DECODER,
        PA_ENCODER
        PREFIX_ENCODER,
    };
    enum KernelType : int {
```

```
        KERNELTYPE_DEFAULT = 0,
        KERNELTYPE_HIGH_PRECISION
    };
    enum ClampType : int {
        CLAMP_TYPE_UNDEFINED = 0,
        CLAMP_TYPE_MIN_MAX
    };
    enum MaskType : int {
        MASK_TYPE_UNDEFINED = 0,
        MASK_TYPE_NORM,
        MASK_TYPE_ALIBI,
        MASK_TYPE_NORM_COMPRESS,
        MASK_TYPE_ALIBI_COMPRESS,
        MASK_TYPE_ALIBI_COMPRESS_SQRT,
        MASK_TYPE_ALIBI_COMPRESS_LEFT_ALIGN,
        MASK_TYPE_SLIDING_WINDOW_NORM,
        MASK_TYPE_SLIDING_WINDOW_COMPRESS
    };
    enum KvCacheCfg :int {
        K_CACHE_V_CACHE = 0,
        K_BYPASS_V_BYPASS,
    };
    enum ScaleType : int {
        SCALE_TYPE_TOR = 0,
        SCALE_TYPE_LOGN,
        SCALE_TYPE_MAX
    };
    enum QuantType : int {
        TYPE_QUANT_UNDEFINED = 0,
        TYPE_QUANT_UNQUANT = 0,
        TYPE_DEQUANT_FUSION,
        TYPE_QUANT_QKV_OFFLINE,
        TYPE_QUANT_QKV_ONLINE
    };
    enum CacheType : int8_t {
        CACHE_TYPE_NORM = 0,
        CACHE_TYPE_SWA = 1
    };
    QuantType quantType = TYPE_QUANT_UNQUANT;
    aclDataType outDataType = ACL_DT_UNDEFINED;
    int32_t headNum = 0;
    int32_t kvHeadNum = 0;
    float qScale = 1;
    float qkScale = 1;
    bool batchRunStatusEnable = false;
    uint32_t isTriuMask = 0;
    CalcType calcType = UNDEFINED;
    KernelType kernelType = KERNELTYPE_DEFAULT;
    ClampType clampType = CLAMP_TYPE_UNDEFINED;
    float clampMin = 0;
    float clampMax = 0;
    MaskType maskType = MASK_TYPE_UNDEFINED;
    KvCacheCfg kvcacheCfg = K_CACHE_V_CACHE;
    ScaleType scaleType = SCALE_TYPE_TOR;
    InputLayout inputLayout = TYPE_BSND;
    uint32_t mlaVHeadSize = 0;
    CacheType cacheType = CACHE_TYPE_NORM;
    uint32_t windowSize = 0;
    uint8_t rsv[64] = {0};
};
```

10.59.6 参数列表

在使用时应带有命名空间atb::infer::SelfAttentionParam的参数，成员描述如下：

成员名称	类型	默认值	描述	是否必选
quantType	QuantType	TYPE_QUANT_UNQUANT	quant类型。	否
outDataType	aclDataType	ACL_DT_UNDEFINED	output数据类型。默认值表示根据输入tensors自动推导输出tensors数据类型。	否
headNum	int32_t	0	query头大小, 需大于0	是
kvHeadNum	int32_t	0	kv头数量，该值需要用户根据使用的模型实际情况传入。	是
qScale	float	1	query缩放系数。	否
qkScale	float	1	算子tor值, 在Q*K^T后乘。	是
batchRunStatusEnable	bool	false	是否开启动态batch。	否
isTriuMask	uint32_t	0	是否开启倒三角优化, 只有mask为倒三角的时候才能开启优化。	否
calcType	CalcType	UNDEFINED	计算类型。	是
kernelType	KernelType	KERNELTYPE_DEFAULT	内核精度类型。	否
clampType	ClampType	CLAMP_TYPE_UNDEFINED	clamp类型。	否
clampMin	float	0	clamp功能最小值。	否
clampMax	float	0	clamp功能最大值。	否
maskType	MaskType	MASK_TYPE_UNDEFINED	mask类型。	否
kvcacheCfg	KvCacheCfg	K_CACHE_V_CACHE	kvcache配置。	否
scaleType	ScaleType	SCALE_TYPE_TOR	scale类型。	否
inputLayout	InputLayout	TYPE_BSND	数据排布格式默认为BSND。	否
mlaVHeadSize	uint32_t	0	大于0时开启MLA合并kvcache功能，表示kv合并传入时v的head_size取值范围为[0,576]。	否

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述	是否必选
windowSize	0	uint32_t	SWA 窗口大小。需合理设置窗口大小，若其大于q/k 的maxSeqlen会导致SWA退化成普通SelfAttention操作。	否
cacheType	CACHE_TYPE_NORM	CacheType	cache内部排布类型, 为CACHE_TYPE_SWA开启SWA KVCache优化，只储存后windowSize个token的KVCache， 控制KVCache的长度不超过windowSize, 以此减少显存占用。	否
rsv[64]	uint8_t	{0}	预留参数。	否

上表中类型为自定义类型的，其描述如下：

- **QuantType**：表示quant类型的枚举量，其具体取值如下。
  - TYPE_QUANT_UNDEFINED：默认值，不与量化融合，此时q、k、v为bf16/float16。
  - TYPE_QUANT_UNQUANT：默认值，不与量化融合，此时q、k、v为bf16/float16。
  - TYPE_DEQUANT_FUSION：与反量化融合, 预留类型，当前不能够取此值。
  - TYPE_QUANT_QKV_OFFLINE：离线INT8量化，只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
  - TYPE_QUANT_QKV_ONLINE：在线INT8量化，只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- **outDataType**：output数据类型，其具体取值如下。
  - 只支持PA_ENCODER，且“QuantType”不为TYPE_QUANT_UNQUANT（格式为aclDataType）。
- **kvHeadNum**：表示kv头数量， 该值需要用户根据使用的模型实际情况传入，其具体取值如下。
  - kvHeadNum = 0时，keyCache的k_head_num，valueCache的v_head_num与query的num_heads一致，均为num_heads的数值。
  - kvHeadNum != 0时，keyCache的k_head_num，valueCache的v_head_num与kvHeadNum值相同。
- **CalcType**：表示计算类型，其具体取值如下。
  - UNDEFINED：同时使用flashAttention的decoder和encoder。
  - ENCODER：使用FlashAttention的encoder。
  - DECODER：使用FlashAttention的decoder。

 版权所有 © 华为技术有限公司

  - PA_ENCODER：使用PagedAttention的encoder。
  - PREFIX_ENCODER：用于FA支持prefix cache场景。
- **KernelType**：表示内核精度类型，其具体取值如下。
  - KERNELTYPE_DEFAULT ： 输入float16，batchmatmul使用float16，输出float16。
  - KERNELTYPE_HIGH_PRECISION ： 输入float16，batch matmul使用float，输出float16。
- **ClampType**：表示clamp类型，其具体取值如下。
  - CLAMP_TYPE_UNDEFINED：不进行缩放。
  - CLAMP_TYPE_MIN_MAX：进行缩放，同时指定最大最小值。
- **MaskType**：表示mask类型，其具体取值如下。
  - MASK_TYPE_UNDEFINED：默认值，全0mask。
  - MASK_TYPE_NORM：倒三角mask。
  - MASK_TYPE_ALIBI：Alibi Mask。
  - MASK_TYPE_NORM_COMPRESS：倒三角压缩mask。
  - MASK_TYPE_ALIBI_COMPRESS：alibi压缩mask。
  - MASK_TYPE_ALIBI_COMPRESS_SQRT：alibi压缩开平方mask。
  - MASK_TYPE_ALIBI_COMPRESS_LEFT_ALIGN：alibi压缩mask左对齐。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
  - MASK_TYPE_SLIDING_WINDOW_NORM： sliding window attention mask。
  - MASK_TYPE_SLIDING_WINDOW_COMPRESS：sliding window attention压缩mask。
- **KvCacheCfg**：表示kvcache配置，不支持“calcType”为PA_ENCODER，其具体取值如下。
  - K_CACHE_V_CACHE ： 默认值，进行kvcache处理。
  - K_BYPASS_V_BYPASS：直接传入kvcache。
- **ScaleType**：表示scale类型，其具体取值如下。
  - SCALE_TYPE_TOR：默认值，不开启LogN缩放。
  - SCALE_TYPE_LOGN：注意力使用LogN缩放，quantType只能是0。
  - SCALE_TYPE_MAX：边界值，仅用于判断是否出界。
- **InputLayout**：输入数据排布格式，InputLayout类型枚举值，其具体取值如下。
  - 默认为TYPE_BSND，此时枚举值为0。
  - 输入数据排布格式为TYPE_BNSD时，枚举值为1。
- **CacheType**：cache内部排布类型，其具体取值如下。
  - CACHE_TYPE_NORM：正常cache。
  - CACHE_TYPE_SWA：固定长度cache。

 版权所有 © 华为技术有限公司

10.59.7 输入输出列表

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
query	[ntokens, qHiddenSize]	float16/bf16	ND	npu	query矩阵。	基础场景
key	[ntokens, kHiddenSize]	float16/bf16	ND	npu	key矩阵，当“kvcacheCfg”配置为K_BYPASS_V_BYPASS，不传入。	基础场景
value	[ntokens, vHiddenSize]	float16/bf16	ND	npu	value矩阵，当“kvcacheCfg”配置为K_BYPASS_V_BYPASS，不传入。	基础场景
cacheK	[layerNum, batch, maxSeqLen, kHiddenSize]	float16/bf16	Atlas 800I A2推理产品/Atlas A2训练系列产品：ND Atlas推理系列产品：NZ	cpu/npu	• NPU：存储之前所有的k，本次执行时将key刷新到cacheK上 。 • CPU：输入为已经准备好的cacheK，输入时根据分成batch个tensor作为std::vector<tensor>传入，此时layer维度要求为1。	基础场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
cacheV	[layerNum, batch, maxSeqLen, vhiddenSize]	float16/bf16	Atlas 800I A2推理产品/Atlas A2训练系列产品：ND Atlas推理系列产品：NZ	npu/cpu	• NPU：存储之前所有的v，本次执行时将value刷新到cacheV上。 • CPU：输入为已经准备好的cacheV，输入时根据分成batch个tensor作为std::vector<tensor>传入，此时layer维度要求为1。	基础场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
attentionMask	[maxSeqLen, maxSeqLen]/[batch, maxSeqLen, maxSeqLen]/[batch, 1, maxSeqLen]/[batch, headNum, maxSeqLen, maxSeqLen]/[512,512]	float16/bf16	Atlas 800 I A2推理产品/Atlas A2训练系列产品：ND Atlas推理系列产品：ND/NZ	npu	五种shape分别对应： 1.所有batch相同或SWA norm mask，方阵。 2.batch不同时的方阵。 3. batch不同时的向量。 4. alibi场景。 5.SWA compress mask。	mask场景
tokenOffset	[batch]	int32/uint32	ND	cpu	计算完成后的token偏移。	基础场景
seqLen	[batch]	int32/uint32	ND	cpu	等于1时，为增量或全量；大于1时，为全量。	基础场景
layerId	[1]	int32/uint32	ND	npu	取cache的kv中哪一个kv进行计算。	基础场景
batchStatus	[batch]	int32/uint32	ND	cpu	开启控制可计算batch功能时，通过标志位控制具体需要运算的batch。	控制可计算batch场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
logN	• 全量阶段为[maxSeqLen] • 增量阶段为[batch]	float	ND	npu	全量阶段为长度maxSeqLen的logN序列，batch内每条请求根据自己的序列长度seqlen从该向量中取值；增量阶段为各batch新增请求对应的logN；当scale类型为logN时需要传此tensor。	LogN场景
output	[nTokens, vHiddenSize]	float16/bf16	ND	npu	输出。	基础场景

当参数“calcType”为PA_ENCODER时，输入输出如下：

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
query	[nTokens, headNum, headSize]	float16/bf16 开启量化功能时为int8	ND	npu	query矩阵, nTokens情况较复杂，见**10.59.8 规格约束**第一项。	基础场景
key	[nTokens, headNum, headSize]	float16/bf16 开启量化功能时为int8	ND	npu	key矩阵。	基础场景
value	[nTokens, headNum, headSize]	float16/bf16 开启量化功能时为int8	ND	npu	value矩阵。当mlaVHeadSize > 0时不传此tensor。	基础场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
mask	[maxSeqLen, maxSeqLen]/[batch, maxSeqLen, maxSeqLen]/[batch, 1, maxSeqLen]/[batch, headNum, maxSeqLen, maxSeqLen] 开启mask压缩功能时与FA有所不同，见**10.59.9.6 压缩mask**	float16/bf16	Atlas 800I A2推理产品/Atlas A2训练系列产品：ND Atlas推理系列产品：NZ	npu	四种shape分别对应： 1.所有batch相同，方阵。 2. batch不同时的方阵。 3. decoder场景下，q_seqlen为1时的mask向量。 4. alibi场景。 当maskType为undefined时不传此tensor。	mask场景
seqLen	[batch]	int32	ND	cpu	等于1时，为增量或全量；大于1时，为全量。	基础场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
slopes	[headNum]	• Atlas 800I A2 推理产品/Atlas A2 训练系列产品：float • Atlas 推理系列产品：float16	ND	npu	当“maskType”为alibi压缩时需传入此tensor，为alibi mask每个head的系数。	alibi压缩mask场景
logN	全量阶段为[maxSeqLen]	float	ND	npu	全量阶段为长度maxSeqLen的logN序列，batch内每条请求根据自己的序列长度seqlen从该向量中取值；当scale类型为logN时需要传此tensor。	LogN场景
output	[nTokens, headNum, headSize]	float16/bf16	ND	npu	输出。	基础场景

当参数“calcType”为PREFIX_ENCODER时，输入输出如下：

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
query	[batch*qSeqLen, qHiddenSize] 或 [batch*qSeqLen, headNum, headSize]	float16/bf16	ND	npu	query矩阵。 • query的shape和key、value的不需要一致。 • PrefixCacheLen的计算由kvseqlen减seqLen得到，由于实现方案限制，需要让prefixCacheLen 128对齐。 • headSize需要16对齐。	基础场景
key	[numBlocks, blockSize, kvHiddenSize] 或 [numBlocks, blockSize, headNum, headSize]	float16/bf16	ND	npu	key矩阵。 • key，value的HeadSize <= 128。 • blockSize必须设为128。 • BlockSize × HeadSize <= 128 × 128。	基础场景
value	[numBlocks, blockSize, kvHiddenSize] 或 [numBlocks, blockSize, headNum, headSize]	float16/bf16	ND	npu	value矩阵。key，value的shape必须一致。	基础场景
blockTables	[batch, maxBlockNum]	int32	ND	npu	每个query的kvcache的block table，第一维是token索引，第二维表示block索引。 blocktable 中item的value < maxBlockNum max(kvSeqLen) / blockSize <= maxblockNum	基础场景

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	cpu/npu	描述	使用场景
mask	1. [256, 256] 2. [head, seqLen, 128] 3. [128, 128]	float16/bf16	ND	npu	仅支持压缩mask。 • MASK_TYPE_ALIBI_COMPRESS支持维度1和2。 • MASK_TYPE_ALIBI_COMPRESS_SQRT支持维度1和2。 • MASK_TYPE_NORM_COMPRESS支持维度3。	基础场景
seqLen	[batch]	int32/uint32	ND	cpu	query对应的每个batch的序列长度。	基础场景
kvSeqLen	[batch]	int32/uint32	ND	cpu	key, value对应的每个batch的序列长度。	基础场景
slopes	[head]	float32	ND	npu	即alibi coefficient，当使用alibiMask时传入tensor时，为alibiMask的每个head的系数。	alibi压缩mask场景
output	[batch*qSeqLen, qHiddenSize]或[batch*qSeqLen, headNum, headSize]	float16/bf16	ND	npu	输出。维度和query一致。	基础场景

10.59.8 规格约束

- 输入数据排布格式为TYPE_BSND时：
  - 在非PA_ENCODER下，Atlas 800I A2 推理产品/Atlas A2 训练系列产品上的query，key，value可传二维[nTokens, hiddenSize]或四维[batch, seqLen, headNum, headSize]。
  - 在PA_ENCODER下，Atlas 800I A2 推理产品/Atlas A2 训练系列产品上的query，key，value可传二维[nTokens, hiddenSize]或三维[nTokens, headNum, headSize]。
- Atlas 推理系列产品上：
  - 0<batch<=2000。
  - mask大小必须使用真实的maxSeqLen。
  - qSeqLen必须等于kvSeqLen。

 版权所有 © 华为技术有限公司

- 当在以下场景时，headSize = vHeadSize，范围为（0，256]，其他情况headSize可以不等于vHeadSize，二者的范围为（0，576]。
  - 开启量化。
  - 注意力使用logN缩放特性。
  - “inputLayout”为TYPE_BNSD。
  - 在Atlas 推理系列产品上运行，此时headSize必须为16的倍数。
  - 在Atlas 训练系列产品上运行，此时headSize必须为16的倍数。
- 关于tensor维度中的nTokens：
  - Atlas 800I A2 推理产品/Atlas A2 训练系列产品上为各batch上seqLen之和。
  - Atlas 推理系列产品上：PA_ENCODER下为所有batch的seqLen之和向上对齐到16的整数倍。
  - 其余情况下为所有batch上的seqLen先向上对齐到16的整数倍，再求和。
- 关于选取何种mask：
  - 在全量阶段，依据原生模型的配置，选择“maskType”是MASK_TYPE_NORM或MASK_TYPE_ALIBI。
  - 在增量阶段，如果原模型非alibi，且mask没有做padding，可选择“maskType”为UNDEFINED，即不传mask；其他情况则依据原生模型的配置，选择“maskType”是MASK_TYPE_NORM或MASK_TYPE_ALIBI。
  - 在长序列场景下，可以考虑压缩mask，压缩mask的构造方法详见**压缩mask**。
- 在PA_ENCODER下：
  - qScale不生效。
  - query、key、value的维度数要相同。
- 若想使用GQA模式，需满足headNum > kvHeadNum且headNum%kvHeadNum == 0。
- Atlas 训练系列产品只支持PA_ENCODER，仅支持基础功能，不支持**10.59.9 功能列表**中的所有功能。
- 在CalcType为ENCODER，DECODER且非bypass场景下hiddensize必须为16的倍数。
- query、key、value的headSize为非64的倍数时，会导致算子性能下降，为了确保最佳性能，建议将query、key、value的headSize设置为64的倍数。
- cpu侧的tensor需要利用**hostData**设置数据的地址，无需设置deviceData；npu侧的tensor需要利用**deviceData**设置数据的地址，无需设置hostData。特殊情况：ENCODER/DECODER在非kv-bypass场景下tokenOffset和Seqlen这两个tensor如果都同时传了deviceData和hostData，则会采用deviceData中的数据进行kvcache部分的计算。

10.59.9 功能列表

10.59.9.1 各功能共存情况

Y表示两功能不冲突，N表示两功能冲突。

calctype 为 ENCODER 或 DECODER

	控制可计算batch	高精度	clamp缩放	压缩mask	kv-bypass	logN缩放	BNSD维度输入	kv tensorlist格式输入	SWA
控制可计算batch	-	Y	Y	Y	Y	Y	Y	Y	N
高精度	-	-	Y	Y	Y	Y	Y	Y	N
clamp缩放	-	-	-	Y	Y	Y	Y	Y	N
压缩mask	-	-	-	-	Y	Y	Y	Y	Y
kv-bypass	-	-	-	-	-	Y	Y	Y	Y
logN缩放	-	-	-	-	-	-	N	N	N
BNSD维度输入	-	-	-	-	-	-	-	N	N
kv tensorlist格式输入	-	-	-	-	-	-	-	-	N
SWA	-	-	-	-	-	-	-	-	-

calctype 为 PA_ENCODER

	高精度	clamp缩放	压缩mask	logN缩放	qkv全量化	BNSD维度输入	MLA合并输入kvcache功能	SWA
高精度	-	Y	Y	Y	Y	N	Y	N
clamp缩放	-	-	Y	Y	Y	Y	Y	N
压缩mask	-	-	-	Y	Y	N	N	Y
logN缩放	-	-	-	-	N	N	N	N
qkv全量化	-	-	-	-	-	N	Y	N
BNSD维度输入	-	-	-	-	-	-	N	N
MLA合并输入kvcache功能	-	-	-	-	-	-	-	N
SWA	-	-	-	-	-	-	-	-

10.59.9.2 控制可计算 batch

功能

指定某几个batch参与attention计算。

开启方式

参数“batchRunStatusEnable”置为true，并传入batchRunStatus作为输入tensor。

batchRunStatus为0，1组成的tensor。0代表该位置的batch不参与计算，1代表参与计算。

 版权所有 © 华为技术有限公司

特殊约束

- 不支持Atlas 推理系列产品。
- 开启此功能时输入的cacheK, cacheV的维度为[batch, maxSeqLen, hiddenSize]。
- 不支持PA_ENCODER。
- 不支持PREFIX_ENCODER。

10.59.9.3 全量/增量分离

功能

当模型接入paged attention时，全量阶段选用SelfAttention且“calcType”置为PA_ENCODER，增量阶段选用PagedAttention。而当模型使用传统flash attention时，增量与全量阶段都使用SelfAttention；此时为提高计算效率，可在全量与增量阶段选用不同的calcType。

开启方式

全量阶段：参数“calcType”置为ENCODER。

增量阶段：参数“calcType”置为DECODER。

特殊约束

无

10.59.9.4 高精度

功能

在进行attention计算时，Q•KT的结果有可能溢出float16，导致算子输出tensor中间出现NAN值；此时可开启此功能，算子内部使用float32承载中间结果。

开启方式

参数“kernelType”置为KERNELTYPE_HIGH_PRECISION。

特殊约束

开启此功能时，传入的mask中需把-inf换成1。

高精度功能只在Atlas 800I A2 推理产品/Atlas A2 训练系列产品上才能生效。

10.59.9.5 clamp 缩放

功能

相当于对传入的q做torch.clamp。

开启方式

参数“clampType”置为CLAMP_TYPE_MIN_MAX。

 版权所有 © 华为技术有限公司

传入参数“clampMin”与参数“clampMax”以给出clamp的上下界。

特殊约束

不支持Atlas 推理系列产品。

10.59.9.6 压缩 mask

功能

又名mask-free。

在长序列场景下，由于seqLen较大，需要的mask的大小也会变大。此时可开启此功能，传入压缩后的mask，以减小显存占用。

开启方式

参数“isTriuMask”置为1。

- 在rope场景下：将变量“maskType”置为MASK_TYPE_NORM_COMPRESS。
- 在alibi场景下：依据具体需要，将变量“maskType”置为MASK_TYPE_ALIBI_COMPRESS或MASK_TYPE_ALIBI_COMPRESS_SQRT或MASK_TYPE_ALIBI_COMPRESS_LEFT_ALIGN。

alibi压缩mask场景需传入slopes。

特殊约束

- alibi压缩mask只有“calcType”置为PA_ENCODER和PREFIX_ENCODER时生效。
  - “maskType”为MASK_TYPE_ALIBI_COMPRESS或MASK_TYPE_ALIBI_COMPRESS_SQRT时，mask的维度：

    在Atlas 800I A2 推理产品/Atlas A2 训练系列产品上为[headNum, seqLen, 128]或[256, 256]。

    在Atlas 推理系列产品上为[headNum,128//16,maxSeqLen,16]或[1,256//16,256,16]。
  - “maskType”为MASK_TYPE_NORM_COMPRESS时，mask的维度：

    在Atlas 800I A2 推理产品/Atlas A2 训练系列产品上为[128,128]。

    在Atlas 推理系列产品上为[1,128//16,128,16]。
- 使用alibi压缩mask时，qkv的headSize要小于等于128。
- MASK_TYPE_ALIBI_COMPRESS_LEFT_ALIGN仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。此时mask的维度为[256, 256]。
  - Atlas 800I A2 推理产品/Atlas A2 训练系列产品上，当数据类型为float16时，alibi压缩mask只有开启高精度才有效。
- 不同压缩mask的构造方法：
  - **对于norm场景：**

    对应maskType为MASK_TYPE_NORM_COMPRESS。

    mask为128 * 128的倒三角，其中：

    i. **当为float16场景时，mask如下图所示。**

 版权所有 © 华为技术有限公司

图 10-62 float16 场景下的压缩 norm mask

0	-inf	-inf	-inf	-inf	-inf	-inf
0	0	-inf	-inf	-inf	-inf	-inf
0	0	0	-inf	-inf	-inf	-inf
0	0	0	0	-inf	-inf	-inf
0	0	0	0	0	-inf	-inf
0	0	0	0	0	0	-inf
0	0	0	0	0	0	0

ii. **当为bf16/float16高精度场景时，mask如下图所示。**

图 10-63 bf16/float16 场景下的压缩 norm mask

0	1	1	1	1	1	1
0	0	1	1	1	1	1
0	0	0	1	1	1	1
0	0	0	0	1	1	1
0	0	0	0	0	1	1
0	0	0	0	0	0	1
0	0	0	0	0	0	0

- **对于alibi场景：**

  alibi mask可以拆解为

  $$alibiMask = alibiBias * alibiCoefficient + triangularMask$$

  其中，alibi coefficient为每个head各不相同的系数，triangularMask代表倒三角mask。压缩mask的场景下，输入tensor中的mask即为压缩后的alibi bias，slopes即为alibi coefficient。针对alibi bias的压缩有如下三种情况：

  i. **右对齐alibi bias**

  对应“maskType”为MASK_TYPE_ALIBI_COMPRESS。

  如下图所示，为512 * 512的压缩前的alibi bias。

 版权所有 © 华为技术有限公司

图 10-64 512 * 512 压缩前的 alibi bias

0							
-1	0						
-2	-1	0					
-3	-2	-1	0				
-4	-3	-2	-1	0			
-5	-4	-3	-2	-1	0		
-6	-5	-4	-3	-2	-1	0	
-7	-6	-5	-4	-3	-2	-1	0

对应的压缩后的256 * 256的压缩后的alibi bias如下图所示。

图 10-65 256 * 256 压缩后的 alibi bias

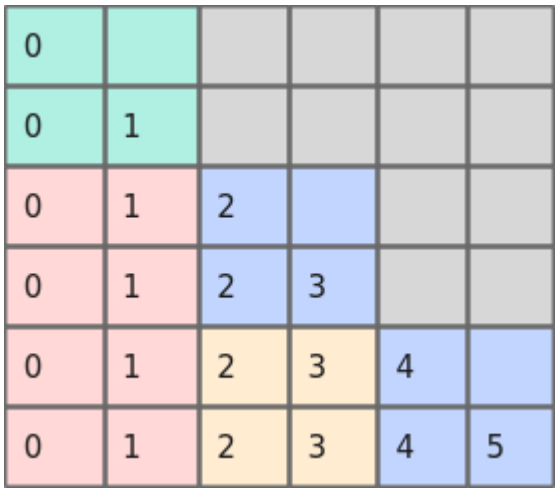
0			
-1	0		
-2	-1	0	
-3	-2	-1	0

ii. **开平方的右对齐alibi bias**
对应“maskType”为MASK_TYPE_ALIBI_COMPRESS_SQRT。
在右对齐alibi bias的基础上对于bias本身做了开根号处理，i的bias变为$-\sqrt{1}$，$-\sqrt{2}$...，其余与i一致。

iii. **左对齐的alibi bias**
对应“maskType”为MASK_TYPE_ALIBI_COMPRESS_LEFT_ALIGN。
压缩前的alibi bias如下图所示。

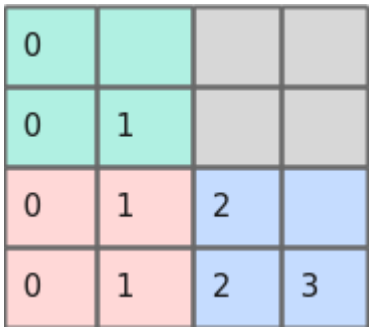
 版权所有 © 华为技术有限公司

图 10-66 压缩前的 alibi bias



压缩后的alibi bias如下图所示。

图 10-67 压缩后的 alibi bias



注意：左对齐的压缩mask只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

10.59.9.7 kv-bypass

功能

在某些场景下，客户想自行管理kvCache的过程，不想使用加速库做把kv刷新到kvCache的动作。

开启方式

参数“kvCacheCfg”置为K_BYPASS_V_BYPASS。同时输入tensor不传k、v。

特殊约束

此功能只有“calcType”为非PA_ENCODER、非PREFIX_ENCODER时生效。

 版权所有 © 华为技术有限公司

10.59.9.8 logN 缩放

功能

在一般的情况下，上文计算公式中的Zoom函数，其缩放系数由参数qkScale给出。当用户想要使用logN形式的缩放函数时，可开启此功能。

开启方式

参数“scaleType”置为SCALE_TYPE_LOGN。

传入logN作为输入tensor。

“kernelType”置为KERNELTYPE_HIGH_PRECISION。

特殊约束

- 开启logN功能，“scaleType”需为SCALE_TYPE_LOGN，“calcType”需为DECODER或PA_ENCODER，分别对应增量阶段和全量阶段；Atlas 800I A2 推理产品/Atlas A2 训练系列产品上“calcType”为PA_ENCODER时额外需要“kernelType”为KERNELTYPE_HIGH_PRECISION。
- logN功能与量化场景不支持同时开启。
- 当“inputLayout”为TYPE_BNSD时，“ScaleType”必须为SCALE_TYPE_TOR（不支持LogN缩放）。

10.59.9.9 qkv 全量化

功能

支持量化好的q, k, v传入，降低显存占用。

开启方式

参数“quantType”置为TYPE_QUANT_QKV_OFFLINE或TYPE_QUANT_QKV_ONLINE, 分别为离线量化与在线量化。

在线全量化需要添加四个输入tensor，分别是qkDescale、qkOffset、vpvDescale、vpvOffset，当采用离线量化时，需要在在线全量化新增的输入tensor的基础上额外传入pScale作为输入tensor。

若干输入tensor需要传入指定的维度，如下表所示：

参数	维度	数据类型	格式	npu or cpu	描述
query	[nTokens, headNum, headSize]	int8	ND	npu	query矩阵。
key	[nTokens, headNum, headSize]	int8	ND	npu	key矩阵。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	npu or cpu	描述
value	[nTokens, headNum, headSize]	int8	ND	npu	value矩阵。
mask	同**10.45.8.2 mask类型**，开启mask压缩功能时与FA有所不同，见**压缩mask**	float16/ bf16	ND/NZ	npu	四种shape分别对应： 1.所有batch相同，方阵。 2. batch不同时的方阵。 3. batch不同时的向量。 4. alibi场景。 当maskType为undefined时不传此tensor。
seqLen	[batch]	int32	ND	cpu	等于1时，为增量或全量；大于1时，为全量。
slopes	[headNum]	Atlas 800I A2 推理产品/Atlas A2 训练系列产品：float16	ND	npu	当maskType为alibi压缩，且mask为[256,256]时需传入此tensor，为alibi mask每个head的系数。
qkDescale	[headNum]	float	ND	npu	为Q*K^T的反量化scale参数。
qkOffset	[headNum]	int32	ND	npu	作为Q*K^T的反量化offset参数。预留tensor，需传任意非空tensor，实际暂未使用。
vpvDescale	[headNum]	float	ND	npu	quantType=2时，为P*V的反量化scale参数；quantType=3时，为V的反量化scale参数。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	npu or cpu	描述
vpvOffset	[headNum]	int32	ND	npu	仅全量化场景传此tensor（即quantType=2或3）。quantType=2时，为P*V的反量化offset参数；quantType=3时，为V的反量化offset参数。 预留tensor，需传任意非空tensor，实际暂未使用。
pScale	[headNum]	float	ND	npu	P的离线量化scale参数，当开启离线全量化时需要传此tensor（即quantType=2）。当开启在线全量化时不传此tensor（即quantType=3）。
output	[nTokens, headNum, headSize]	float16/bf16	ND	npu	输出。

其中pScale仅在离线量化场景传入，在线量化场景则不传此输入tensor。

特殊约束

使用全量化时需要指定输出tensor的数据类型，具体为使用参数outDataType，该参数只能是ACL_FLOAT16或ACL_BF16。

“calctype”须为PA_ENCODER。

不支持Atlas 推理系列产品。

10.59.9.10 BNSD 维度输入

功能

一般的，传入SelfAttention算子的q，k，v的维度为[batch, seqLen, headNum, head_dim]，即[b, s, n, d]，或者是它合轴后的变种。在某些场景下，传入[b, n, s, d]性能更好。

开启方式

参数“inputLayout”置为TYPE_BNSD。

输入参数如下：

 版权所有 © 华为技术有限公司

- “calctype”不为PA_ENCODER时，在Atlas 800I A2 推理产品/Atlas A2 训练系列产品上：

输入tensor	维度	数据类型	格式	描述
query	[batch, headNum, seqLen, headSize]	float16/bf16	ND	query矩阵。
cacheK	[layer, batch, headNum, seqLen, headSize]	float16/bf16	ND	• NPU：存储之前所有的k，本次执行时将key刷新到cacheK上 。 • CPU：输入为已经准备好的cacheK，输入时根据分成batch个tensor作为std::vector<tensor>传入，此时layer维度要求为1。
cacheV	[layer, batch, headNum, seqLen, headSize]	float16/bf16	ND	• NPU：存储之前所有的v，本次执行时将value刷新到cacheV上。 • CPU：输入为已经准备好的cacheV，输入时根据分成batch个tensor作为std::vector<tensor>传入，此时layer维度要求为1。

- “calctype”不为PA_ENCODER时，在Atlas 推理系列产品上：

输入tensor	维度	数据类型	格式	描述
query	[batch, headNum, seqLen, headSize]	float16/bf16	ND	query矩阵。
cacheK	[layer, batch*headNum, headSize / 16, kvMaxSeq, 16]	float16/bf16	NZ	• NPU：存储之前所有的k，本次执行时将key刷新到cacheK上 。 • CPU：输入为已经准备好的cacheK，输入时根据分成batch个tensor作为std::vector<tensor>传入，此时layer维度要求为1。

 版权所有 © 华为技术有限公司

输入tensor	维度	数据类型	格式	描述
cacheV	[layer, batch*headNum, headSize / 16, kvMaxSeq, 16]	float16/ bf16	NZ	• NPU：存储之前所有的v，本次执行时将value刷新到cacheV上。 • CPU：输入为已经准备好的cacheV，输入时根据分成batch个tensor作为std::vector<tensor>传入，此时layer维度要求为1。

- “calctype”为PA_ENCODER时：

参数	维度	数据类型	格式	描述
query	[batch, headNum, qSeqLen, headSize]	bf16	ND	query矩阵。
cacheK	[batch, kvHeadNum, kvSeqLen, headSize]	bf16	ND	key矩阵。
cacheV	[batch, kvHeadNum, kvSeqLen, headSize]	bf16	ND	value矩阵。
SeqLen	[batch] / [2, batch]	int32/ uint32	ND	若shape为[batch] ，代表每个batch的序列长度，query，cacheK，cacheV相同。 若shape为[2，batch]，SeqLen[0]代表query的序列长度，SeqLen[1]代表cacheK，cacheV的序列长度。
attnOut	[batch, headNum, qSeqLen, headSize]	bf16	ND	输出tensor。

特殊约束

- BNSD只有开启kv-bypass功能，即参数“kvcacheCfg”置为K_BYPASS_V_BYPASS或“calctype”置为PA_ENCODER时才可用。
- 使用BNSD维度输入且“calctype”不为PA_ENCODER时，“maskType”不能为MASK_TYPE_UNDEFINED。“calctype”为PA_ENCODER时，“maskType”只能为MASK_TYPE_UNDEFINED。

 版权所有 © 华为技术有限公司

- seqlen的dimNum只有“calctype”为PA_ENCODER且开启BNSD之后才能为2。
- “calctype”为PA_ENCODER时不支持Atlas 推理系列产品。
- query、key、value的HeadSize必须相同且<=256。
- “calctype”为PA_ENCODER时使用BNSD维度输入功能，不支持高精度。

10.59.9.11 kv tensorlist 格式输入

功能

支持以tensorlist，即二级指针结构的形式而非整个tensor作为cacheK与cacheV的输入。

开启方式

如何构造该场景中的variantPack:

atb Tensor定义：

```
struct Tensor {
    //! \brief Tensor描述信息
    TensorDesc desc;
    //! \brief TensorNPU内存地址。
    void *deviceData = nullptr;
    //! \brief TensorCPU内存地址。
    void *hostData = nullptr;
    //! \brief “deviceData”或“hostData”指向内容的内存大小。
    uint64_t dataSize = 0;
};
```

variantPack中的输入tensors，在需要使能tensor list时，将对应tensor的deviceData置为nullptr，将hostData指向构造的tensorList即可。

特殊约束

此功能只在开启kv-bypass功能（即参数“kvcacheCfg”置为K_BYPASS_V_BYPASS），且“calcType”为DECODER时生效。

10.59.9.12 MLA 合并输入 kvCache 功能

功能

过去attention算子会将k和v存储在显存中，随着模型越来越大，输入序列越来越长，kvCache也越来越大，对显存的容量造成很大压力，造成性能上的瓶颈。因此Multi-Head Latent Attention提出将kvCache压缩成一个一个较小的向量，代替原来的kvCache输入进fa算子进行计算。

开启方式

- calcType= PA_ENCODER
- mlaVHeadSize > 0

 版权所有 © 华为技术有限公司

特殊约束

- MLA合并输入kvCache功能不支持Atlas 推理系列产品，不支持alibi mask，多头自适应压缩（rope 场景），多头自适应压缩（alibi 场景），logN缩放，BNSD输入排布。
- 开启MLA合并输入kvCache功能后key和value的headSize范围为（0, 576]。
- mlaVHeadSize不能大于query和key的headSize。
- 开启MLA合并输入kvCache功能后query，key，value可传二维[nTokens, hiddenSize]或三维[nTokens, headNum, headSize]或四维[batch, seqLen, headNum, headSize]，且query，key的维度数可以不同。
- MLA合并kvCache功能支持全量化场景，一起开启时query和key的headSize范围为（0, 576]，mlaVHeadSize不能大于query和key的headSize。

10.59.9.13 Sliding window attention（SWA）功能

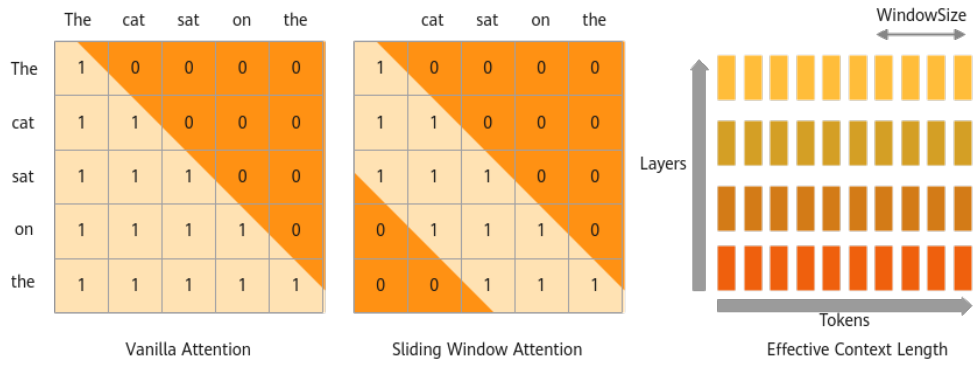
功能

Mistral 7B，Attention部分在GQA（Group Query Attention）的基础上，叠加了SWA（Sliding Window Attention）的优化，可以进一步提高推理速度，并降低显存。

当前FA算子在进行推理时采用的是取所有KV进行计算，可以应用SWA提升推理速度，越远距离的信息，对当前位置的重要性越低，对于距离超过窗口大小的两个token不参与注意力分数计算。

通过设计mask实现n>m或m-n>windowLen（n为kvId, m为qId）时，注意力分数跳过计算。如图10-68，若windowSize = 3，“The”将不会计算与“the”之间的注意力分数。

图 10-68 Sliding Window Attention 的 mask 功能示意



Decoder支持仅长度为windowLen的最新KV历史参与计算。

kvCache优化：比如窗口大小=4，则当第5个token需要缓存，直接替换掉第1个token，这样就可以保持kv缓存有一个最大值（为窗口大小），而不会无限增长。

开启方式

- windowSize > 0
- maskType必须为MASK_TYPE_SLIDING_WINDOW_NORM或MASK_TYPE_SLIDING_WINDOW_COMPRESS

 版权所有 © 华为技术有限公司

特殊约束

- 开启特性后“cacheType”可以为CACHE_TYPE_NORM 或 CACHE_TYPE_SWA, 不开启特性“cacheType”只能为 CACHE_TYPE_NORM。
- Sliding Window Attention特性不支持动态batch，高精度，clamp缩放，qkv全量化，mla，logN缩放特性，BNSD数据排布。
- Sliding Window Attention特性在calcType=DECODER场景下，“maskType”不能为MASK_TYPE_SLIDING_WINDOW_COMPRESS，且不传入attentionMask。
- 开启特性必须满足以下两个条件。如果只满足一个条件，会校验报错。
  - windowSize > 0。
  - maskType必须为MASK_TYPE_SLIDING_WINDOW_NORM或MASK_TYPE_SLIDING_WINDOW_COMPRESS。

SWA mask 样例生成参考

当windowSize >= seqlen时，mask和不开启SWA时一样，为上三角，否则mask生成可参考如下：

- **maskType=MASK_TYPE_SLIDING_WINDOW_NORM, dtype为float16。**
  以 max(seqLen)为5，windowSize为3为例。

  图 10-69 maskType=MASK_TYPE_SLIDING_WINDOW_NORM, dtype 为 float16 时的 mask 样例

0.0	-65536	-65536	-65536	-65536
0.0	0.0	-65536	-65536	-65536
0.0	0.0	0.0	-65536	-65536
-65536	0.0	0.0	0.0	-65536
-65536	-65536	0.0	0.0	0.0

python代码示例：

 版权所有 © 华为技术有限公司

```
swa_mask = np.ones(shape=[maxseq,maxseq]) * -65536.0
triu_mask = np.triu(swa_mask, 1)
tril_mask = np.tril(swa_mask, -window_size)
swa_mask = triu_mask + tril_mask
```

C++代码示例：

```
static constexpr uint32_t  SWA_MASK_SIZE = maxSeq;
std::vector<float> create_attention_mask(uint32_t windowSize, uint32_t embeddim) {
    std::vector<float> attention_mask(SWA_MASK_SIZE * SWA_MASK_SIZE, -65536.0);

    for (uint32_t i = 0; i < SWA_MASK_SIZE; ++i) {
        uint32_t offset = i >= windowSize ? (i - windowSize + 1) : 0;
        for (uint32_t j = offset; j < i + 1; ++j) {
            attention_mask[i * SWA_COMPRESS_MASK_SIZE + j] = 0.0;
        }
    }
    return attention_mask;
}
```

- **maskType=MASK_TYPE_SLIDING_WINDOW_NORM, dtype为bf16。**

  以 max(seqLen)为5，windowSize为3为例。

  图 10-70 maskType=MASK_TYPE_SLIDING_WINDOW_NORM, dtype 为 bf16 时的 mask 样例

0.0	1.0	1.0	1.0	1.0
0.0	0.0	1.0	1.0	1.0
0.0	0.0	0.0	1.0	1.0
1.0	0.0	0.0	0.0	1.0
1.0	1.0	0.0	0.0	0.0

python代码示例：

```
swa_mask = np.ones(shape=[maxseq,maxseq]) * 1.0
triu_mask = np.triu(swa_mask, 1)
tril_mask = np.tril(swa_mask, -window_size)
swa_mask = triu_mask + tril_mask
```

C++代码示例：

 版权所有 © 华为技术有限公司

```
static constexpr uint32_t  SWA_MASK_SIZE = maxSeq;
std::vector<float> create_attention_mask(uint32_t windowSize, uint32_t embeddim) {
    std::vector<float> attention_mask(SWA_MASK_SIZE * SWA_MASK_SIZE, 1.0);

    for (uint32_t i = 0; i < SWA_MASK_SIZE; ++i) {
        uint32_t offset = i >= windowSize ? (i - windowSize + 1) : 0;
        for (uint32_t j = offset; j < i + 1; ++j) {
            attention_mask[i * SWA_COMPRESS_MASK_SIZE + j] = 0.0;
        }
    }
    return attention_mask;
}
```

- **maskType=MASK_TYPE_SLIDING_WINDOW_COMPRESS, dtype为float16。**

  python代码示例：

```
swa_mask = np.ones(shape=(1, 512, 512)) * -65536.0
pp_n = 128 if head_size <= 128 else 64 # head_size为每个注意力头的嵌入向量的大小
if window_size <= pp_n * 3:
    true_size = window_size
else:
    if window_size % pp_n == 0:
        true_size = pp_n * 3
    else:
        true_size = pp_n * 2 + window_size % pp_n
triu_mask = np.triu(swa_mask, 1)
tril_mask = np.tril(swa_mask, -true_size)
swa_mask = triu_mask + tril_mask
```

  C++代码示例：

```
static constexpr uint32_t  SWA_COMPRESS_MASK_SIZE = 512;
std::vector<float> create_attention_mask(uint32_t windowSize, uint32_t embeddim) {
    std::vector<float> attention_mask(SWA_COMPRESS_MASK_SIZE * SWA_COMPRESS_MASK_SIZE,
-65536.0);
    uint32_t blockSize = embeddim > 128 ? (16384 / embeddim / 16 * 16) : 128;
    uint32_t compressWindow = windowSize > 3 * blockSize ? (2 * blockSize + windowSize %
blockSize) : windowSize;
    for (uint32_t i = 0; i < SWA_COMPRESS_MASK_SIZE; ++i) {
        uint32_t offset = i >= compressWindow ? (i - compressWindow + 1) : 0;
        for (uint32_t j = offset; j < i + 1; ++j) {
            attention_mask[i * SWA_COMPRESS_MASK_SIZE + j] = 0.0;
        }
    }
    return attention_mask;
}
```

- **maskType=MASK_TYPE_SLIDING_WINDOW_COMPRESS, dtype为bf16。**

  python代码示例：

```
swa_mask = np.ones(shape=(1, 512, 512)) * 1
pp_n = 128 if head_size <= 128 else 64 # head_size为每个注意力头的嵌入向量的大小
if window_size <= pp_n * 3:
    true_size = window_size
else:
    if window_size % pp_n == 0:
        true_size = pp_n * 3
    else:
        true_size = pp_n * 2 + window_size % pp_n
triu_mask = np.triu(swa_mask, 1)
tril_mask = np.tril(swa_mask, -true_size)
swa_mask = triu_mask + tril_mask
```

  C++代码示例：

```
static constexpr uint32_t  SWA_COMPRESS_MASK_SIZE = 512;
std::vector<float> create_attention_mask(uint32_t windowSize, uint32_t embeddim) {
    std::vector<float> attention_mask(SWA_COMPRESS_MASK_SIZE * SWA_COMPRESS_MASK_SIZE,
1.0);
    uint32_t blockSize = embeddim > 128 ? (16384 / embeddim / 16 * 16) : 128;
    uint32_t compressWindow = windowSize > 3 * blockSize ? (2 * blockSize + windowSize %
blockSize) : windowSize;
```

 版权所有 © 华为技术有限公司

```
    for (uint32_t i = 0; i < SWA_COMPRESS_MASK_SIZE; ++i) {
        uint32_t offset = i >= compressWindow ? (i - compressWindow + 1) : 0;
        for (uint32_t j = offset; j < i + 1; ++j) {
            attention_mask[i * SWA_COMPRESS_MASK_SIZE + j] = 0.0;
        }
    }
    return attention_mask;
}
```

10.59.9.14 FA 支持 Prefix Encoder

功能

FA支持按block存放的kvCache。BlockTables维度为[batch, maxBlockNum]，其中每个batch对应存放当前query对应的key和value的index，maxBlockNum约为key，value中最长kvSeqLen，kvSeqLen/blockSize向上取整。

开启方式

calcType置为PREFIX_ENCODER。

特殊约束

- 不支持qScale，需设为1。
- isTriuMask需设置为1。
- kernelType必须设为KERNELTYPE_HIGH_PRECISION，因为kv计算内部均使用float32计算。
- 不支持clamp，clampMin、clampMax需设置为0。
- inputLayout只支持TYPE_BSND。
- 不支持mla，mlaVHeadSize需设为0。
- PrefixCacheLen的计算由kvseqlen减seqLen得到，由于实现方案限制，需要让prefixCacheLen 128对齐。
- headNum有如下约束：
  - headNum > 0
  - headNum >= kvHeadNum
  - headNum是kvHeadNum的整数倍
- maskType支持MASK_TYPE_ALIBI_COMPRESS、MASK_TYPE_ALIBI_COMPRESS_SQRT或MASK_TYPE_NORM_COMPRESS。

10.59.10 算子调用示例（C++）

10.59.10.1 fa_encoder_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：FA Encoder场景。

```
#include <iostream>
#include <vector>
#include <numeric>
```

 版权所有 © 华为技术有限公司

```
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const uint32_t BATCH_SIZE = 100;                // 批处理大小
std::vector<int32_t> seqLenHost(BATCH_SIZE, 16);  // host侧tensor值，用于存储每个批处理中的序列长度
std::vector<int32_t> tokenOffsetHost(BATCH_SIZE, 16);                    // host侧tensor值，token偏移
std::vector<int32_t> layerId(1, 0);                                      // device侧，kvCache中取哪个计算
const uint32_t NTOKENS = accumulate(seqLenHost.begin(), seqLenHost.end(), 0);  // sum(seqLenHost)
const uint32_t MAX_SEQ_LEN = 1024;                                       // 最大序列长度
const uint32_t HEAD_NUM = 32;                                           // 头数
const uint32_t KV_HEAD_NUM = 32;                                         // kv头数
const uint32_t HEAD_SIZE = 64;                                          // 头大小

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。序列长度向量，等于1时，为增量或全量；大于1时，为全量
 * @param tokenOffsetHost host侧tensor。计算完成后的token偏移
 * @param layerId layerId，取cache的kv中哪一个kv进行计算
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream,
    std::vector<int32_t> &seqLenHost, std::vector<int32_t> &tokenOffsetHost, std::vector<int32_t> &layerId)
{
    uint32_t qHiddenSize = HEAD_NUM * HEAD_SIZE;
    uint32_t kvHiddenSize = KV_HEAD_NUM * HEAD_SIZE;

    // 创建query tensor
    std::vector<float> qData(NTOKENS * qHiddenSize, 1.0);
    atb::Tensor tensorQ = CreateTensorFromVector(
        contextPtr, stream, qData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, qHiddenSize});
    // 创建key，value tensor
    std::vector<float> kvData(NTOKENS * kvHiddenSize, 1.0);
    atb::Tensor tensorK = CreateTensorFromVector(
        contextPtr, stream, kvData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, qHiddenSize});
    atb::Tensor tensorV = CreateTensorFromVector(
        contextPtr, stream, kvData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, qHiddenSize});
    std::vector<float> kvCacheData(BATCH_SIZE * MAX_SEQ_LEN * kvHiddenSize, 1.0);
    std::vector<int64_t> kvCacheShape = {1, BATCH_SIZE, MAX_SEQ_LEN, kvHiddenSize};
    atb::Tensor tensorCacheK =
        CreateTensorFromVector(contextPtr, stream, kvCacheData, ACL_FLOAT16,
aclFormat::ACL_FORMAT_ND, kvCacheShape);
    atb::Tensor tensorCacheV =
        CreateTensorFromVector(contextPtr, stream, kvCacheData, ACL_FLOAT16,
aclFormat::ACL_FORMAT_ND, kvCacheShape);
    // 创建norm mask，值为-inf的上三角mask
    std::vector<float> maskData(BATCH_SIZE * MAX_SEQ_LEN * MAX_SEQ_LEN, 0);
    for (int i = 0; i < BATCH_SIZE; ++i) {
        for (int j = 0; j < MAX_SEQ_LEN; ++j) {
            for (int k = j + 1; k < MAX_SEQ_LEN; ++k) {
                maskData[i * MAX_SEQ_LEN * MAX_SEQ_LEN + j * MAX_SEQ_LEN + k] = -32768;
            }
        }
    }
    atb::Tensor tensorMask = CreateTensorFromVector(
        contextPtr, stream, maskData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE,
MAX_SEQ_LEN, MAX_SEQ_LEN});
    // 创建tokenOffset，host侧tensor
    atb::Tensor tensorTokenOffset = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    tensorTokenOffset.hostData = tokenOffsetHost.data();  // host侧tensor，拷贝值
    // 创建seqLen，host侧tensor
    atb::Tensor tensorSeqLen = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    tensorSeqLen.hostData = seqLenHost.data();  // host侧tensor，拷贝值
```

 版权所有 © 华为技术有限公司

```
    // 创建layerId
    atb::Tensor tensorLayerId = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {1});
    CHECK_STATUS(aclrtMemcpy(tensorLayerId.deviceData,
        tensorLayerId.dataSize,
        layerId.data(),
        sizeof(short) * layerId.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {tensorQ,
        tensorK,
        tensorV,
        tensorCacheK,
        tensorCacheV,
        tensorMask,
        tensorTokenOffset,
        tensorSeqLen,
        tensorLayerId};
    return inTensors;
}

/**
 * @brief 创建一个FA encoder的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::SelfAttentionParam opParam;
    opParam.maskType = atb::infer::SelfAttentionParam::MaskType::MASK_TYPE_NORM;
    opParam.headNum = HEAD_NUM;
    opParam.kvHeadNum = KV_HEAD_NUM;
    opParam.calcType = atb::infer::SelfAttentionParam::CalcType::ENCODER;
    atb::Operation *encoderOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(opParam, &encoderOp));
    return encoderOp;
}

int main(int argc, char **argv)
{
    // kv隐藏层大小，用于输出tensor shape
    uint32_t kvHiddenSize = KV_HEAD_NUM * HEAD_SIZE;

    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // FA Encoder示例
    atb::Operation *encoderOp = PrepareOperation();
    // 准备输入张量
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream, seqLenHost, tokenOffsetHost, layerId);  // 放入
输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS,
kvHiddenSize});
    variantPack.outTensors.push_back(tensorOut);  // 放入输出tensor
    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(encoderOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // FA Encoder执行
```

 版权所有 © 华为技术有限公司

```
    encoderOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    CHECK_STATUS(aclrtFree(tensorOut.deviceData));
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(encoderOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "FA Encoder demo success!" << std::endl;
    return 0;
}
```

10.59.10.2 fa_encoder_inference_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。本用例仅支持Atlas 推理系列产品。

与示例1相较本示例主要有以下修改点：

- kvCache shape改为：[block_num, head_size * head_num / 16, block_size, 16]。
- kvCache数据格式改为：ACL_FORMAT_FRACTAL_NZ。

场景：FA Encoder场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const uint32_t BATCH_SIZE = 1;                  // 批处理大小
std::vector<int32_t> seqLenHost(BATCH_SIZE, 16);  // host侧tensor值，用于存储每个批处理中的序列长度
std::vector<int32_t> tokenOffsetHost(BATCH_SIZE, 16);                    // host侧tensor值，token偏移
std::vector<int32_t> layerId(1, 0);                                      // device侧，kvCache中取哪个计算
const uint32_t NTOKENS = accumulate(seqLenHost.begin(), seqLenHost.end(), 0);  // sum(seqLenHost)
const uint32_t MAX_SEQ_LEN = 256;                                        // 最大序列长度
const uint32_t HEAD_NUM = 16;                                            // 头数
const uint32_t KV_HEAD_NUM = 16;                                         // kv头数
const uint32_t HEAD_SIZE = 16;                                           // 头大小
const uint32_t LAYER_NUM = 1;                                            // 层大小

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。序列长度向量，等于1时，为增量或全量；大于1时，为全量
 * @param tokenOffsetHost host侧tensor。计算完成后的token偏移
 * @param layerId layerId，取cache的kv中哪一个kv进行计算
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream,
    std::vector<int32_t> &seqLenHost, std::vector<int32_t> &tokenOffsetHost, std::vector<int32_t> &layerId)
{
    uint32_t qHiddenSize = HEAD_NUM * HEAD_SIZE;
    uint32_t kvHiddenSize = KV_HEAD_NUM * HEAD_SIZE;
```

 版权所有 © 华为技术有限公司

```
    // 创建query tensor
    std::vector<float> qData(NTOKENS * qHiddenSize, 1.0);
    atb::Tensor tensorQ = CreateTensorFromVector(
        contextPtr, stream, qData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, qHiddenSize});
    // 创建key，value tensor
    std::vector<float> kvData(NTOKENS * kvHiddenSize, 1.0);
    atb::Tensor tensorK = CreateTensorFromVector(
        contextPtr, stream, kvData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, qHiddenSize});
    atb::Tensor tensorV = CreateTensorFromVector(
        contextPtr, stream, kvData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, qHiddenSize});
    std::vector<unsigned int16_t> kvCacheData(LAYER_NUM * BATCH_SIZE * MAX_SEQ_LEN * kvHiddenSize,
0x3C00);
    std::vector<int64_t> kvCacheShape = {LAYER_NUM, BATCH_SIZE, kvHiddenSize / 16, MAX_SEQ_LEN, 16};
    atb::Tensor tensorCacheK = CreateTensorFromVector(
        contextPtr, stream, kvCacheData, ACL_FLOAT16, aclFormat::ACL_FORMAT_FRACTAL_NZ,
kvCacheShape, ACL_FLOAT16);
    atb::Tensor tensorCacheV = CreateTensorFromVector(
        contextPtr, stream, kvCacheData, ACL_FLOAT16, aclFormat::ACL_FORMAT_FRACTAL_NZ,
kvCacheShape, ACL_FLOAT16);
    // 创建tokenOffset，host侧tensor
    atb::Tensor tensorTokenOffset = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    tensorTokenOffset.hostData = tokenOffsetHost.data();  // host侧tensor，拷贝值
    // 创建seqLen，host侧tensor
    atb::Tensor tensorSeqLen = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    tensorSeqLen.hostData = seqLenHost.data();  // host侧tensor，拷贝值
    // 创建layerId
    atb::Tensor tensorLayerId = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {1});
    CHECK_STATUS(aclrtMemcpy(tensorLayerId.deviceData,
        tensorLayerId.dataSize,
        layerId.data(),
        sizeof(short) * layerId.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {
        tensorQ, tensorK, tensorV, tensorCacheK, tensorCacheV, tensorTokenOffset, tensorSeqLen,
tensorLayerId};
    return inTensors;
}

/**
 * @brief 创建一个FA encoder的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::SelfAttentionParam opParam;
    opParam.headNum = HEAD_NUM;
    opParam.kvHeadNum = KV_HEAD_NUM;
    opParam.calcType = atb::infer::SelfAttentionParam::CalcType::ENCODER;
    atb::Operation *encoderOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(opParam, &encoderOp));
    return encoderOp;
}

int main(int argc, char **argv)
{
    // kv隐藏层大小，用于输出tensor shape
    uint32_t kvHiddenSize = KV_HEAD_NUM * HEAD_SIZE;

    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);
```

 版权所有 © 华为技术有限公司

```
    // FA Encoder示例
    atb::Operation *encoderOp = PrepareOperation();
    // 准备输入张量
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream, seqLenHost, tokenOffsetHost, layerId);  // 放入
输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS,
kvHiddenSize});
    variantPack.outTensors.push_back(tensorOut);  // 放入输出tensor
    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(encoderOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // FA Encoder执行
    encoderOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    CHECK_STATUS(aclrtFree(tensorOut.deviceData));
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(encoderOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "FA Encoder demo success!" << std::endl;
    return 0;
}
```

10.59.10.3 fa_pa_encoder_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：FA PA_Encoder场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const uint32_t BATCH_SIZE = 1;                    // 批处理大小
std::vector<int32_t> seqLenHost(BATCH_SIZE, 16);  // host侧tensor值，用于存储每个批处理中的序列长度
const uint32_t NTOKENS = accumulate(seqLenHost.begin(), seqLenHost.end(), 0);  // sum(seqLenHost)
const uint32_t MAX_SEQ_LEN = 1024;                                             // 最大序列长度
const uint32_t HEAD_NUM = 32;                                                  // 头数
const uint32_t KV_HEAD_NUM = 32;                                               // kv头数
const uint32_t HEAD_SIZE = 64;                                                 // 头大小

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。序列长度向量，等于1时，为增量或全量；大于1时，为全量
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
```

 版权所有 © 华为技术有限公司

```
 */
atb::SVector<atb::Tensor> PrepareInTensor(
    atb::Context *contextPtr, aclrtStream stream, std::vector<int32_t> &seqLenHost)
{
    // 创建query tensor
    atb::Tensor tensorQ = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>(NTOKENS * HEAD_NUM * HEAD_SIZE, 1.0),
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {NTOKENS, HEAD_NUM, HEAD_SIZE});
    // 创建key，value tensor
    std::vector<float> kvData(NTOKENS * KV_HEAD_NUM * HEAD_SIZE, 1.0);
    atb::Tensor tensorK = CreateTensorFromVector(
        contextPtr, stream, kvData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, KV_HEAD_NUM,
HEAD_SIZE});
    atb::Tensor tensorV = CreateTensorFromVector(
        contextPtr, stream, kvData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS, KV_HEAD_NUM,
HEAD_SIZE});
    std::vector<float> maskData(BATCH_SIZE * MAX_SEQ_LEN * MAX_SEQ_LEN, 0);
    // 创建norm mask，值为-inf的上三角mask
    for (int i = 0; i < BATCH_SIZE; ++i) {
        for (int j = 0; j < MAX_SEQ_LEN; ++j) {
            for (int k = j + 1; k < MAX_SEQ_LEN; ++k) {
                maskData[i * MAX_SEQ_LEN * MAX_SEQ_LEN + j * MAX_SEQ_LEN + k] = -32768;
            }
        }
    }
    atb::Tensor tensorMask = CreateTensorFromVector(
        contextPtr, stream, maskData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE,
MAX_SEQ_LEN, MAX_SEQ_LEN});
    // 创建seqLen，host侧tensor
    atb::Tensor tensorSeqLen = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    tensorSeqLen.hostData = seqLenHost.data();  // seqLenHost中的值为seqLen
    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {tensorQ, tensorK, tensorV, tensorMask, tensorSeqLen};
    return inTensors;
}

/**
 * @brief 创建一个FA encoder的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::SelfAttentionParam paOpParam;
    paOpParam.maskType = atb::infer::SelfAttentionParam::MaskType::MASK_TYPE_NORM;
    paOpParam.headNum = HEAD_NUM;
    paOpParam.kvHeadNum = KV_HEAD_NUM;
    paOpParam.calcType = atb::infer::SelfAttentionParam::CalcType::PA_ENCODER;
    atb::Operation *paEncoderOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(paOpParam, &paEncoderOp));
    return paEncoderOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // FA PAEncoder示例
    atb::Operation *paEncoderOp = PrepareOperation();
```

 版权所有 © 华为技术有限公司

```
    // 准备输入张量
    atb::VariantPack paVariantPack;
    paVariantPack.inTensors = PrepareInTensor(context, stream, seqLenHost);  // 放入输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS,
KV_HEAD_NUM, HEAD_SIZE});
    paVariantPack.outTensors.push_back(tensorOut);  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(paEncoderOp->Setup(paVariantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // FA Encoder执行
    paEncoderOp->Execute(paVariantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    CHECK_STATUS(aclrtFree(tensorOut.deviceData));
    for (atb::Tensor &inTensor : paVariantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(paEncoderOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "FA PA Encoder demo success!" << std::endl;
    return 0;
}
```

10.59.10.4 fa_prefix_encoder_demo.cpp

前置条件和编译命令请参见**4 算子调用示例**。当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

场景：FA Prefix_Encoder场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const uint32_t BATCH_SIZE = 4;                      // 批处理大小
std::vector<int32_t> seqLenHost = {16, 16, 32, 32};  // host侧tensor值，用于存储Query每个批处理中的序列长度
const uint32_t NTOKENS = accumulate(seqLenHost.begin(), seqLenHost.end(), 0);  // sum(seqLenHost)
std::vector<int32_t> kvSeqLenHost = {16, 144, 32, 288};  // host侧tensor值，用于存储Key, Value每个批处理中的序列长度
const uint32_t NUM_BLOCKS = accumulate(kvSeqLenHost.begin(), kvSeqLenHost.end(), 0);  // sum(kvSeqLenHost)
const uint32_t HEAD_NUM = 32;                                                  // query头数
const uint32_t KV_HEAD_NUM = 32;                                                 // kv头数
const uint32_t HEAD_SIZE = 64;                                                // 头大小
const uint32_t BLOCK_SIZE = 128;                                               // 以block存放的kv块大小

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。Query序列长度向量
```

 版权所有 © 华为技术有限公司

```
 * @param kvSeqLenHost host侧tensor。Key, Value序列长度向量
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(
    atb::Context *contextPtr, aclrtStream stream, std::vector<int32_t> &seqLenHost, std::vector<int32_t>
&kvSeqLenHost)
{
    // 创建query tensor
    atb::Tensor tensorQ = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>(NTOKENS * HEAD_NUM * HEAD_SIZE, 1.0),
        ACL_FLOAT16,
        aclFormat::ACL_FORMAT_ND,
        {NTOKENS, HEAD_NUM, HEAD_SIZE});
    // 创建key，value tensor
    std::vector<float> kvData(NUM_BLOCKS * BLOCK_SIZE * KV_HEAD_NUM * HEAD_SIZE, 1.0);
    std::vector<int64_t> kvShape = {NUM_BLOCKS, BLOCK_SIZE, KV_HEAD_NUM, HEAD_SIZE};
    atb::Tensor tensorK =
        CreateTensorFromVector(contextPtr, stream, kvData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
kvShape);
    atb::Tensor tensorV =
        CreateTensorFromVector(contextPtr, stream, kvData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
kvShape);
    // 创建blockTables
    atb::Tensor tensorBlockTables = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE, 4});
    std::vector<int32_t> blockTablesData(16);
    std::iota(blockTablesData.begin(), blockTablesData.end(), 0);
    CHECK_STATUS(aclrtMemcpy(tensorBlockTables.deviceData,
        tensorBlockTables.dataSize,
        blockTablesData.data(),
        sizeof(int32_t) * blockTablesData.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    // 创建alibi128mask，值为1的上三角
    std::vector<float> maskData = std::vector<float>(HEAD_NUM * NTOKENS * 128, 0);  // alibi128 mask
    for (int i = 0; i < HEAD_NUM; ++i) {
        for (int j = 0; j < NTOKENS; ++j) {
            for (int k = j + 1; k < 128; ++k) {
                maskData[i * NTOKENS * 128 + j * 128 + k] = 1;
            }
        }
    }
    atb::Tensor tensorMask = CreateTensorFromVector(
        contextPtr, stream, maskData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {HEAD_NUM, NTOKENS,
128});
    // 创建seqLen，host侧tensor
    atb::Tensor tensorSeqLen = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    tensorSeqLen.hostData = seqLenHost.data();
    // 创建kvSeqLen，host侧tensor
    atb::Tensor tensorKvSeqLen = CreateTensor(ACL_INT32, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE});
    tensorKvSeqLen.hostData = kvSeqLenHost.data();
    atb::Tensor tensorSlopes = CreateTensor(ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {HEAD_SIZE});
    std::vector<float> slData(HEAD_SIZE, 1.0);
    CHECK_STATUS(aclrtMemcpy(tensorSlopes.deviceData,
        tensorSlopes.dataSize,
        slData.data(),
        sizeof(float) * slData.size(),
        ACL_MEMCPY_HOST_TO_DEVICE));
    atb::SVector<atb::Tensor> inTensors = {
        tensorQ, tensorK, tensorV, tensorBlockTables, tensorMask, tensorSeqLen, tensorKvSeqLen,
tensorSlopes};
    return inTensors;
}

/**
 * @brief 创建一个FA encoder的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
```

 版权所有 © 华为技术有限公司

```
{
    atb::infer::SelfAttentionParam prefixOpParam;
    prefixOpParam.headNum = HEAD_NUM;
    prefixOpParam.kvHeadNum = KV_HEAD_NUM;
    prefixOpParam.calcType = atb::infer::SelfAttentionParam::CalcType::PREFIX_ENCODER;
    prefixOpParam.kernelType = atb::infer::SelfAttentionParam::KernelType::KERNELTYPE_HIGH_PRECISION;
    prefixOpParam.maskType = atb::infer::SelfAttentionParam::MaskType::MASK_TYPE_ALIBI_COMPRESS;
    prefixOpParam.isTriuMask = 1;
    atb::Operation *prefixEncoderOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(prefixOpParam, &prefixEncoderOp));
    return prefixEncoderOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // FA Prefix Encoder示例
    atb::Operation *prefixEncoderOp = PrepareOperation();
    // 准备输入tensor
    atb::VariantPack prefixVariantPack;
    prefixVariantPack.inTensors = PrepareInTensor(context, stream, seqLenHost, kvSeqLenHost);  // 放入输入
tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {NTOKENS,
KV_HEAD_NUM, HEAD_SIZE});
    prefixVariantPack.outTensors = {tensorOut};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspaceSize大小
    CHECK_STATUS(prefixEncoderOp->Setup(prefixVariantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // FA Prefix Encoder执行
    prefixEncoderOp->Execute(prefixVariantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    CHECK_STATUS(aclrtFree(tensorOut.deviceData));
    for (atb::Tensor &inTensor : prefixVariantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(prefixEncoderOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "FA Prefix Encoder demo success!" << std::endl;
    return 0;
}
```

 版权所有 © 华为技术有限公司

10.60 SendOperation

功能

将当前通信卡的输入发送至指定通信卡上。Send和Recv需要配套使用。

图 10-71 Send&Recv 收发数据功能示意图



定义

```
struct SendParam {
    int rank = 0;
    int rankSize = 0;
    int rankRoot = 0;
    uint32_t destRank = 1;
    std::string backend = "hccl";
    HcclComm hcclComm = nullptr;
    CommMode commMode = COMM_MULTI_PROCESS;
    std::string rankTableFile;
    std::string commDomain;
    uint8_t rsv[64] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rank	int	0	当前卡所属通信编号。
rankSize	int	0	通信的卡的数量。
rankRoot	int	0	主通信编号。
destRank	uint32_t	1	通信域内数据接收端的rank编号。
backend	std::string	"hccl"	通信后端指示，仅支持"hccl"。
hcclComm	HcclComm	nullptr	HCCL通信域指针。 默认为空，加速库为用户创建；若用户想要自己管理通信域，则需要传入该通信域指针，加速库使用传入的通信域指针来执行通信算子。
commMode	CommMode	COMM_MULTI_PROCESS	通信模式，**CommMode**类型枚举值。hccl多线程只支持外部传入通信域方式。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
rankTableFile	std::string	-	集群信息的配置文件路径，适用单机以及多机通信场景，当前仅支持hccl后端场景,若单机配置了ranktable，则以ranktable来初始化通信域。 ranktable配置请参见《**TensorFlow 1.15模型迁移指南**》“模型训练 > 执行分布式训练 > 训练执行（配置文件方式设置资源信息） > 准备ranktable资源配置文件”。
commDomain	std::string	-	通信device组用通信域名标识，多通信域时使用，当前仅支持hccl。
rsv[64]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	"hccl": float16/float/int8/int16/int32/int64/bf16	ND	输入tensor。

规格约束

当前只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

rank、rankSize、rankRoot需满足以下条件:

- 0 ≤ rank < rankSize
- 0 ≤ rankRoot < rankSize
- 0 ≤ destRank < rankSize

说明

- 多用户使用时需要使用ATB_SHARE_MEMORY_NAME_SUFFIX环境变量（请参见Transformer加速库环境变量说明）进行共享内存的区分，以进行初始化信息同步。
- 当使用加速库的通信算子异常退出时，需要清空残留数据，避免影响之后的使用，命令参考如下：

```
rm -rf /dev/shm/sem.lccl*
rm -rf /dev/shm/sem.hccl*
ipcrm -a
```

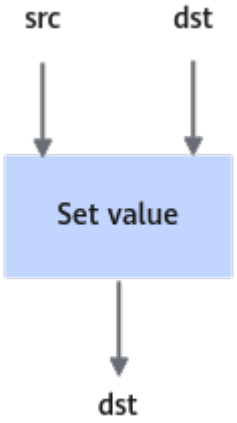
10.61 SetValueOperation

功能

将输入源张量中的内容拷贝到输入目标张量指定位置中。

 版权所有 © 华为技术有限公司

算子上下文



算子功能实现

该拷贝为原地拷贝，最终结果修改在输入目标张量中。

例如：输入目标张量dst: (a,b,c)，输入源张量src: (d,e,f)

dst[starts[0]: ends[0], starts[1]: ends[1], starts[2]: ends[2]] = src

其中，starts[0]: ends[0]长度为src第0维的维度大小，starts[1]: ends[1]长度为src第1维的维度大小，starts[2]: ends[2]长度为src第2维的维度大小。

定义

```
struct SetValueParam {
    SVector<int64_t> starts;
    SVector<int64_t> ends;
    SVector<int64_t> strides;
    uint8_t rsv[8] = {0};
};
```

成员

成员名称	类型	默认值	描述
starts	SVector<int64_t>	-	每一维拷贝起始位置。
ends	SVector<int64_t>	-	每一维拷贝结束位置后一个位置，拷贝到该位置前一个位置为止。
strides	SVector<int64_t>	-	每一维拷贝步长，当前仅支持strides为全1。
rsv[8]	uint8_t	{0}	预留参数。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
dst	[dim_0，dim_1，...，dim_n]	float16/float/int32/int64/bf16	ND	输入tensor1。
src	src的所有维度都小于等于dst对应的维度	与dst一致	ND	输入tensor2。

输出

参数	维度	数据类型	格式	描述
dst	[dim_0，dim_1，...，dim_n]	与src一致	ND	输出tensor。 注：输出dst与同名输入指向同一地址，即进行原地修改。

规格约束

- 输入src和输入dst的维数、数据类型须相同。
- 输入src的各维度大小要求小于或等于输入dst对应维度大小。
- 输入src和输入dst的各维度要求有一个或两个维度不相同，且需要满足：
  - 如果有一个维度不相同，则这个维度不能是最高维（第0维）。
  - 如果有两个维度不相同，则其中一个不同的维度必须是最高维（第0维）。

接口调用示例

```
输入：
starts = {1,2}
ends = {4,4}
strides = {1,1}
dst=[[ 14,  -2,   2, -10, -12],
    [-11,   3,   2, -18,   6],
    [  1, -18, -12,   9,  -6],
    [ 16, -11,   6,  -6,   3],
    [ 12,  18,  14,  10,   9]]
src=[[  1,   0],
    [ -4,   7],
    [  9, -18]]
输出：
dst=[[ 14,  -2,   2, -10, -12],
    [-11,   3,   1,   0,   6],
    [  1, -18,  -4,   7,  -6],
    [ 16, -11,   9, -18,   3],
    [ 12,  18,  14,  10,   9]]
```

 版权所有 © 华为技术有限公司

10.62 SliceOperation

功能

从输入张量某个起始位置中提取指定大小的切片。

算子上下文

图 10-72 SliceOperation 算子上下文



定义

```
struct SliceParam {
    SVector<int64_t> offsets;
    SVector<int64_t> size;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
offsets	SVector<int64_t>	-	每个维度切片的起始位置。 ● 当offsets[i]小于0时（-dimNum<=i<dimNum），例如offsets[i]=-1，输入x的第i维的维度大小为dimNum，则输入x的第i维切片的起始位置为dimNum - 1（即起始位置为最后一维）。 ● 当offsets[i]小于0时，且第i维的维度大小为dimNum，要求dimNum与x之和大于等于0。
size	SVector<int64_t>	-	每个维度切片的大小。 ● 当size[i] = -1时，表示切片的结束位置是第i维的最后一个位置。如果第i维的维度大小为dimNum，则结束位置为dimNum - 1。 ● size中元素要求大于等于-1。且offsets[i] + size[i]须在x的第i维维度的大小范围内。
rsv[8]	uint8_t	{0}	预留参数。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/float/int8/ bool/int32/ uint32/bf16	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
output	和参数size表示的shape一样	float16/float/int8/ bool/int32/ uint32/bf16	ND	输出tensor，维度的大小为size指定的大小。与x的数据类型相同。

规格约束

- 当offsets[i]小于0时，其含义是第i维最高维度开始访问，如offsets= -1，输入x的维度为dimNum，则第i维切片的起始位置为dimNum - 1。
- 当offsets[i]小于0时，且第i维的维度大小为dimNum，要求dimNum与offsets[i]之和大于等于0。
- 当size[i] = -1时，表示切片的结束位置是第i维的最后一个位置。如果第i维的维度大小为dimNum，则结束位置为dimNum - 1。
- size中元素要求大于等于-1。且offsets[i] + size[i]须在x的第i维维度的大小范围内。
- Atlas 200I/500 A2 推理产品上仅支持float16数据类型。

调用接口示例

- **输入**

```
x = [[ -78.9,   78.8,  32.22,   49.7,  82.8, 24.77],
     [  72.2,  20.89,  55.16,   85.5,  72.8,  94.8],
    [ 11.36, -83.56,  65.94, -46.94, 72.44,  68.7]]
offsets = [1, 2]

size = [2, 4]
```

在第0维上，从下标为1开始切片长度为2。

在第1维上从下标为2开始向后切片长度为4（2+4 <= 6）。

- **输出**

```
y.dim = [2, 4]
y = [[ 55.1562,  85.5000,  72.8125,  94.8125],
     [ 65.9375, -46.9375,  72.4375,  68.6875]]
```

若size中的维度为-1, 表示自动推导输出shape。

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

 版权所有 © 华为技术有限公司

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

using namespace atb;
using namespace std;

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @param seqLenHost host侧tensor。序列长度向量，等于1时，为增量或全量；大于1时，为全量
 * @param tokenOffsetHost host侧tensor。计算完成后的token偏移
 * @param layerId layerId，取cache的kv中哪一个kv进行计算
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    uint32_t dim0 = 3;
    uint32_t dim1 = 6;
    // 创建tensor0
    std::vector<float> tensorzero{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18};
    atb::Tensor tensorZero =
        CreateTensorFromVector(contextPtr, stream, tensorzero, ACL_FLOAT, aclFormat::ACL_FORMAT_ND,
{dim0, dim1});

    // 根据顺序将所有输入tensor放入SVector
    atb::SVector<atb::Tensor> inTensors = {tensorZero};
    return inTensors;
}

/**
 * @brief 创建一个slice的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::SliceParam sliceParam;
    sliceParam.offsets = {1, 2};
    sliceParam.size = {2, 4};
    atb::Operation *op = nullptr;
    CHECK_STATUS(atb::CreateOperation(sliceParam, &op));
    return op;
}

int main(int argc, char **argv)
{
    // 1.设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // Slice示例
    atb::Operation *op = PrepareOperation();
    // 准备输入张量
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);                    // 放入输入tensor
```

```
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {2, 4});  // 创建输出
tensor
    variantPack.outTensors.push_back(tensorOut);                                        // 放入输出tensor

    // 计算workspace大小
    uint64_t workspaceSize = 0;
    CHECK_STATUS(op->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }

    // execute阶段
    op->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放内存
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    // 资源释放
    CHECK_STATUS(atb::DestroyOperation(op));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS((aclFinalize()));
    std::cout << "slice demo success!" << std::endl;
    return 0;
}
```

10.63 SoftmaxOperation

功能

Softmax是一种归一化激活函数，可以将Tensor（张量）的某个或多个维度数据归一化，对应维度上每一个元素的范围都在(0, 1)之间并且和为1。

图 10-73 SoftmaxOperation



计算公式

$$\sigma\left(x \right)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for\ j = 1, \ldots, K$$

使用场景

通常作为模型的最后一层，例如在Transformer模型中将Self-Attention的K、Q进行softmax，获取相应的相关性。

 版权所有 © 华为技术有限公司

$$Attention\left(Q, K, V \right) = softmax\left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 200I/500 A2 推理产品	支持	仅支持float16数据类型。
Atlas 推理系列产品	支持	-
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

定义

```
struct SoftmaxParam {
    SVector<int64_t> axes;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
axes	SVector<int64_t>	-	指定轴（维度），axes可以支持多个轴上进行处理。 ● axes不能为空，当指定多个轴时，多个轴之间必须连续且从小到大排列。 ● axes的元素要求大于或等于-1且小于输入x的维度。
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND	输入tensor。

 版权所有 © 华为技术有限公司

输出

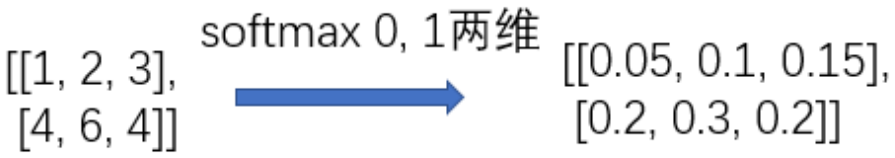
参数	维度	数据类型	格式	描述
output	[dim_0, dim_1, ..., dim_n]	float16/float/bf16	ND	输出tensor。维度和数据类型与输入tensor一致。

功能列表

- **单维度softmax**
  当参数axes只含一个元素时，对输入tensor的对应维度进行softmax操作。当axes=[x]时，等效于Pytorch的

  $$torch.nn.Softmax(dim = x)$$

- **多维度softmax**
  当参数axes含有多个元素，且符合要求时，对输入tensor的对应多个维度进行softmax操作。axes=[x, x+1]时，对x和x+1维度对应的元素进行softmax操作。将axes中连续的多个维度平铺成一维，然后对这一维进行softmax操作，操作完成后再reshape回原来的维度形状。



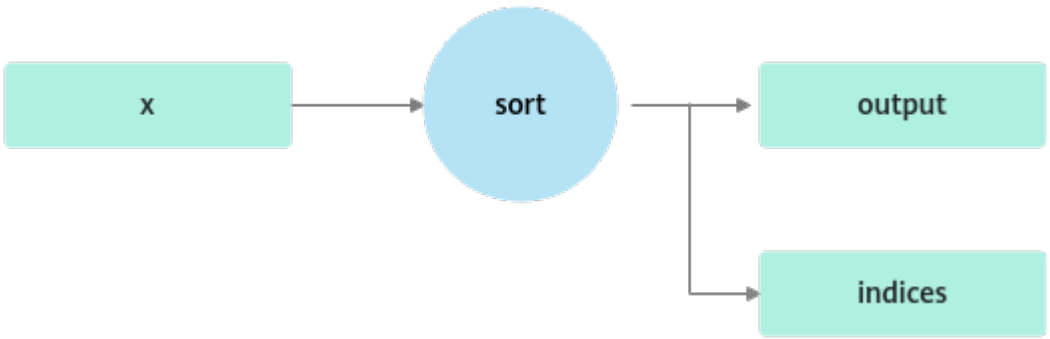
10.64 SortOperation

功能

后处理计算功能。实现输入tensor在最后一维上降序排列，并保留最大的num个元素，输出排序后的tensor及各元素对应的索引。

算子上下文

图 10-74 SortOperation 算子上下文



 版权所有 © 华为技术有限公司

定义

```
struct SortParam {
    SVector<int32_t> num;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
num	SVector<int32_t>	-	排序后保留的最大的元素的数量。 num是一个仅含有一个值的SVector，该值需大于0且小于等于输入x最后一维的大小。
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[dim_0，dim_1，...，dim_n]	float16/bf16/float	ND	最后一维应至少有num个元素。

输出

参数	维度	数据类型	格式	描述
output	[dim_0，dim_1，...，num]	float16/bf16/float	ND	最后一维排序后，最大的num个元素。
indices	[dim_0，dim_1，...，num]	int32	ND	最大的num个元素对应的原索引。

规格约束

- num是一个仅含有一个值的SVector，该值需大于0且小于等于输入x最后一维的大小。
- Atlas 推理系列产品不支持数据类型float。

接口调用示例

输入：

```
num = [1]
x = [[3.0, 4.0],
     [5.0, 6.0]]
```

 版权所有 © 华为技术有限公司

输出：

```
output = [[4.0],
          [6.0]]
indices = [[1.0],
           [1.0]]
```

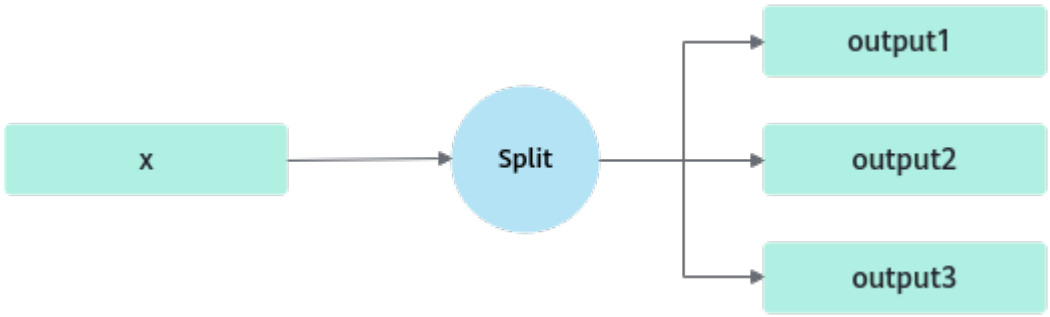
10.65 SplitOperation

功能

将输入张量沿指定维度切分成多个张量。

算子上下文

图 10-75 SplitOperation



算子功能实现

- **参数：**

参数	描述
x	输入的张量，即希望分割的张量。
splitNum	是一个整数，表示等分次数，当前支持2或3。
splitDim	可选，默认0，沿着这个维度进行拆分。默认情况下，沿着第一个维度（即批次维度）进行拆分。

- **返回值：**

  返回若干张量。这些张量是从原张量中根据指定的方式切分出来的。

定义

```
struct SplitParam {
    int32_t splitDim = 0;
    int32_t splitNum = 2;
    SVector<int32_t> splitSizes = {};
    uint8_t rsv[8] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	描述
splitDim	int32_t	0	指定切分的维度索引。 splitDim须位于输入张量x的维度范围内，即如果x的维度为xDim，则splitDim的取值范围为[-xDim, xDim - 1]。 当splitDim为负数时，其含义是从最高维度开始访问，如splitDim = -1，x维度数为dimNum，则拆分维度为dimNum - 1。
splitNum	int32_t	2	等分次数,当前支持2或3。 输入张量x的维度须能够被splitNum整除，且当splitNum = 3时输入x要求是float16或者bf16数据类型。
splitSizes	SVector<int32_t>	-	指定每个输出tensor在切分维度上的大小，不传入此参数时使用等长切分，传入此参数时使用splitV不等长切分。
rsv[8]	uint8_t	{0}	预留参数。

splitNum=2 时输入输出

参数	维度	数据类型	格式	描述
x	[dim_0, ..., dim_splitDim, ..., dim_n]	float16/ int64/ bf16	ND	输入，最高支持8维。
output1	• [dim_0, ⋯, dim_splitDim_1, ..., dim_n] • 当splitSizes为空时，dim_splitDim_1=dim_splitDim/2; • 当splitSizes不为空时，dim_splitDim_2=splitSizes[0]。	float16/ int64/ bf16	ND	输出，切分后的tensor。数据类型与x一致。
output2	• [dim_0, ⋯, dim_splitDim_2, ..., dim_n] • 当splitSizes为空时，dim_splitDim_2=dim_splitDim/2; • 当splitSizes不为空时，dim_splitDim_2=splitSizes[1]。	float16/ int64/ bf16	ND	输出，切分后的tensor。数据类型与x一致。

 版权所有 © 华为技术有限公司

splitNum=3 时输入输出

参数	维度	数据类型	格式	描述
x	[dim_0, ..., dim_splitDim，...，dim_n]	float16/bf16	ND	输入，最高支持8维。
output1	• [dim_0, ⋯, dim_splitDim_1, ..., dim_n] • 当splitSizes为空时，dim_splitDim_1=dim_splitDim/2; • 当splitSizes不为空时，dim_splitDim_1=splitSizes[0]。	float16/bf16	ND	输出，切分后的tensor。数据类型与x一致。
output2	• [dim_0, ⋯, dim_splitDim_2, ..., dim_n] • 当splitSizes为空时，dim_splitDim_2=dim_splitDim/3; • 当splitSizes不为空时，dim_splitDim_2=splitSizes[1]。	float16/bf16	ND	输出，切分后的tensor。数据类型与x一致。
output3	• [dim_0, ⋯, dim_splitDim_3, ..., dim_n] • 当splitSizes为空时，dim_splitDim_3=dim_splitDim/3; • 当splitSizes不为空时，dim_splitDim_3=splitSizes[2]。	float16/bf16	ND	输出，切分后的tensor。数据类型与x一致。

规格约束

- splitDim必须位于输入张量x的维度范围内，即如果x的维度为xDim，则splitDim的取值范围为[-xDim, xDim - 1]。 当splitDim为负数时，其含义是从最高维度开始访问，如splitDim = -1，x维度数为dimNum，则拆分维度为dimNum - 1。使用不等长切分时不支持负数索引。
- splitSizes非空时，维度为splitNum，其每一个元素要求大于等于1。元素之和等于dim_splitDim的大小。
- splitSizes为空时，输入张量x的维度必须能够被splitNum整除。
- 当splitNum = 3时，要求输入x的数据类型是float16或者bfloat16。
- Atlas 200I/500 A2 推理产品上仅支持float16数据类型。

接口调用示例

输入：

 版权所有 © 华为技术有限公司

```
splitDim = 0
splitNum = 3
x = [3, 3, 3, 3, 3, 3, 3, 3, 3]
```

输出：

```
z = [3, 3, 3]
z1 = [3, 3, 3]
z2 = [3, 3, 3]
```

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include <random>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 配置Op参数
    atb::infer::SplitParam opParam;
    opParam.splitDim = 1;         // 设定切分轴为1
    opParam.splitNum = 2;         // 设置切分后得到的块数
    opParam.splitSizes = {2, 3};  // 设置不均匀切分时每块大小

    // 准备VariantPack
    atb::VariantPack variantPack;
    std::vector<int64_t> inputXShape = {1, 5, 2};
    std::vector<int64_t> output1Shape = {1, 2, 2};
    std::vector<int64_t> output2Shape = {1, 3, 2};

    std::vector<float> inTensorXData = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    std::vector<float> output1RefData = {0, 1, 2, 3};
    std::vector<float> output2RefData = {4, 5, 6, 7, 8, 9};

    atb::Tensor inTensorX =
        CreateTensorFromVector(context, stream, inTensorXData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
inputXShape);
    atb::Tensor outTensor1 = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, output1Shape);
    atb::Tensor outTensor2 = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, output2Shape);

    variantPack.inTensors = {inTensorX};
    variantPack.outTensors = {outTensor1, outTensor2};

    // 申请SplitOp
    atb::Operation *splitOp = {nullptr};
    CHECK_STATUS(atb::CreateOperation(opParam, &splitOp));
    uint64_t workspaceSize = 0;
    // ATB Operation 第一阶段接口调用：对输入输出进行检查，并根据需要计算workspace大小
```

 版权所有 © 华为技术有限公司

```
    CHECK_STATUS(splitOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // ATB Operation 第二阶段接口调用：执行算子
    CHECK_STATUS(splitOp->Execute(variantPack, workspacePtr, workspaceSize, context));
    CHECK_STATUS(aclrtSynchronizeStream(stream)); // 流同步，等待device侧任务计算完成

    // 资源释放
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(splitOp));
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));
    CHECK_STATUS(aclFinalize());
    std::cout << "Split demo success!" << std::endl;
    return 0;
}
```

10.66 SwigluQuantOperation

功能

swiglu函数激活对输出进行per token量化。

定义

```
struct SwigluQuantParam{
    enum QuantType:int{
        QUANT_TYPE_PER_TOKEN =0,
    };
    QuantType quantType = QUANT_TYPE_PER_TOKEN;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	是否必选	描述
quantType	QuantType	QUANT_TYPE_PER_TOKEN	[0]	是	PER_TOKEN量化。
rsv[8]	uint8_t	{0}	[0]	否	预留参数。

 版权所有 © 华为技术有限公司

输入

参数	维度	数据类型	格式	描述
inTensor	[nTokens, 2 * hiddenSize]	float16/ bf16	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
outTensor1	[nTokens, hiddenSize]	int8	ND	输出tensor，量化输出。
outTensor2	[nTokens]	float	ND	输出tensor，量化后的scale。

规格约束

- 25*hiddenSize (32Byte对齐后的数字个数)<192KB。
- 仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

接口调用示例（ python ）

```
import sys
import os
import unittest
import torch
import torch_npu
import numpy as np

sys.path.append(os.path.join(os.path.dirname(__file__), "../"))
import operation_test  # NOQA: E402

OP_NAME = "SwigluQuantOperation"
PARAM = {"":1}

class TestSwigluQuantOperation(operation_test.OperationTest):
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def do_swiglu(self, a, b):
        sigmoid_mul_a = self.sigmoid(a) * a
        swiglu_y = sigmoid_mul_a * b
        return swiglu_y

    def do_quant(self, swiglu_y):
        y_tmp = swiglu_y
        y_tmp = np.array(y_tmp)
        y_max = np.amax(np.abs(y_tmp), axis=1)  # 动态量化依赖于每一行的最大值来调整缩放因子
        dynamic_scale_tmp = 127 / y_max
        dynamic_scale = dynamic_scale_tmp.reshape(-1, 1)  # 将一维行向量转换为二维，如(256)转为(256,1)
        y_tmp = y_tmp * dynamic_scale
        quant_y_tmp = np.round(y_tmp)  # 使用 numpy 进行乘法和四舍五入
        quant_y = quant_y_tmp.astype(np.int8)  # 转换为 int8 类型
        dynamic_scale_output = np.array([])
        dynamic_scale = 1 / dynamic_scale
        dynamic_scale_output = np.append(dynamic_scale_output, dynamic_scale)
        return quant_y,dynamic_scale_output
```

 版权所有 © 华为技术有限公司

```
    def SwiGluQuantGolden(self, x_golden):
        a, b = np.split(x_golden, 2, axis=1)
        wiglu_y = self.do_swiglu(a, b)
        quant_y, dynamic_scale = self.do_quant(wiglu_y)
        return torch.from_numpy(quant_y).npu(), torch.from_numpy(dynamic_scale).npu()

    def golden_calc(self, in_tensors):
        golden_res = self.SwiGluQuantGolden(in_tensors[0].cpu())
        return [golden_res[0], golden_res[1]]

    def golden_compare(self, out_tensor, golden_out_tensor):
        actual_output = out_tensor.cpu()
        golden_output = golden_out_tensor.cpu()
        res = False
        if actual_output.dtype == torch.int8:
            diff = np.abs(actual_output - golden_output)
            res = not (diff > 1).nonzero().size(0) > 0
        elif actual_output.dtype == torch.float32:
            res = np.allclose(actual_output, golden_output, rtol=0.0001, atol=0.0001)
        return res

    def test_swi_glu_quant_case0(self):
        input_token_num = 128
        input_hidden_size = 4096

        shape = (input_token_num, input_hidden_size)

        x = torch.empty(shape).uniform_(0, 1).to(torch.float16)
        self.execute(OP_NAME, PARAM, [x.npu().half()])


if __name__ == '__main__':
    unittest.main()
```

10.67 TopkToppSamplingOperation

10.67.1 功能

依据给定的词表概率以及top-k、top-p，从词表概率中选取前k个作为候选，然后选取概率高于top-p的词表按照TopkToppSamplingType的值采取不同的后处理策略，最后选择最合适的词序号以及对应的概率作为输出。

Top-k取样：从tokens里选取k个作为候选，然后根据它们的"likelihood scores"来采样模型从最有可能的k个选项中随机选择一个。

Top-p取样：从tokens里选取累计概率大于或等于概率p的最小集合作为候选，然后随机选择一个。

10.67.2 定义

```
struct TopkToppSamplingParam {
    enum TopkToppSamplingType {
        SAMPLING_UNDEFINED = -1,
        SINGLE_TOPK_SAMPLING,
        BATCH_TOPK_MULTINOMIAL_SAMPLING,
        BATCH_TOPK_EXPONENTIAL_SAMPLING,
        BATCH_TOPK_MULTINOMIAL_LOGPROBS_SAMPLING,
        BATCH_TOPK_EXPONENTIAL_LOGPROBS_SAMPLING,
        SAMPLING_MAX,
    };
    TopkToppSamplingType topkToppSamplingType = SINGLE_TOPK_SAMPLING;
    uint32_t randSeed = 0;
```

 版权所有 © 华为技术有限公司

```
    uint32_t topk = 100;
    std::vector<uint32_t> randSeeds;
    int32_t logProbsSize = 0;
    uint8_t rsv[12] = {0};
};
```

10.67.3 参数列表

成员名称	类型	默认值	描述
topkToppSamplingType	TopkToppSamplingType	SINGLE_TOPK_SAMPLING	取样处理类型。 ● SAMPLING_UNDEFINED：未定义，不作为取样处理类型。 ● SINGLE_TOPK_SAMPLING：非batch级别随机种子、top-k。 ● BATCH_TOPK_MULTINOMIAL_SAMPLING：batch级别随机种子、top-k的multinomial取样。 ● BATCH_TOPK_EXPONENTIAL_SAMPLING：batch级别随机种子，top-k的exponential取样。 ● BATCH_TOPK_MULTINOMIAL_LOGPROBS_SAMPLING：batch级别随机种子、top-k的multinomial取样，输出增加logProbs。仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。 ● BATCH_TOPK_EXPONENTIAL_LOGPROBS_SAMPLING：batch级别随机种子，top-k的exponential取样，输出增加logProbs。仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。 ● SAMPLING_MAX：枚举最大值，不作为取样处理类型。
randSeed	uint32_t	0	top-p阶段随机抽样使用的随机数种子，默认值为0。当topktoppSamplingtype = SINGLE_TOPK_SAMPLING时使用。
topk	uint32_t	100	top-k阶段保留的词的个数，需要小于词表的词数，默认值为100。top-k必须大于0且小于或等于输入probs最后一维的大小。当topktoppSamplingtype = SINGLE_TOPK_SAMPLING时使用。
randSeeds	std::vector<uint32_t>	-	每个batch下top-p阶段随机抽样使用的随机数种子。维度与batch大小一致。当topktoppSamplingtype = BATCH_TOPK_MULTINOMIAL_SAMPLING时使用。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
logProbsSize	int32_t	0	为scalar的int型变量。取值范围是(0, 16384]，在枚举值为BATCH_TOPK_EXPONENTIAL_LOGPROBS_SAMPLING，BATCH_TOPK_MULTINOMIAL_LOGPROBS_SAMPLING时生效。
rsv[12]	uint8_t	{0}	预留参数。

10.67.4 输入输出

topkToppSamplingType = SINGLE_TOPK_SAMPLING（非 batch 级随机种子、topk 取样）

参数	维度	数据类型	格式	描述
probs	[batch, voc_size]	float16/bf16	ND	输入，词表概率。
topp	[batch, 1]	float16/bf16	ND	输入topp，topp截取的概率，batch的值需与probs的一致。
sampled_indices	[batch, 1]	int32	ND	输出，取样的idx。
sampled_probs	[batch, 1]	float16/bf16	ND	输出，取样的值。

topkToppSamplingType = BATCH_TOPK_MULTINOMIAL_SAMPLING（batch 级随机种子，topk 的 multinomial 取样）

参数	维度	数据类型	格式	描述
probs	[batch, voc_size]	float16/bf16	ND	输入，词表概率。
topk	[batch, 1]	int32	ND	输入top-k，topk截取的位置，batch的值需与probs的一致。
topp	[batch, 1]	float16/bf16	ND	输入，topp截取的概率，batch的值需与probs的一致。
sampled_indices	[batch, 1]	int32	ND	输出，取样的idx。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
sampled_probs	[batch, 1]	float16/bf16	ND	输出，取样的值。

topkToppSamplingType = BATCH_TOPK_EXPONENTIAL_SAMPLING（batch 级随机种子，topk 的 Exponential 取样）

参数	维度	数据类型	格式	描述
probs	[batch, voc_size]	float16/bf16	ND	输入，词表概率。
topk	[batch, 1]	int32	ND	输入top-k，topk截取的位置，batch的值需与probs的一致。
topp	[batch, 1]	float16/bf16	ND	输入，topp截取的概率，batch的值需与probs的一致。
exp	[batch, voc_size]	float16/bf16	ND	输入，所除的指数分布，维度需与probs的一致。
sampled_indices	[batch, 1]	int32	ND	输出，取样的idx。
sampled_probs	[batch, 1]	float16/bf16	ND	输出，取样的值。

topkToppSamplingType = BATCH_TOPK_MULTINOMIAL_LOGPROBS_SAMPLING（batch 级随机种子，topk 的 multinomial 取样）

参数	维度	数据类型	格式	描述
probs	[batch, voc_size]	float16/bf16	ND	输入，词表概率。
topk	[batch, 1]	int32	ND	输入top-k，top-k截取的位置，batch的值需与probs的一致。
topp	[batch, 1]	float16/bf16	ND	输入top-p，top-p截取的概率，batch的值需与probs的一致。
rand	[batch, 1]	float	ND	输入，值域为[0, 1]，参考rand() / RAND_MAX。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
sampled_indices	[batch, 1]	int32	ND	输出，取样的序号。
sampled_probs	[batch, 1]	float16/bf16	ND	输出，取样的值。
sampled_logprobs	[batch, logProbsSize]	float	ND	输出，topkToppSamplingType = BATCH_TOPK_MULTINOMIAL_LOGPROBS_SAMPLING时有效。值为对应概率值的对数。

topkToppSamplingType = BATCH_TOPK_EXPONENTIAL_LOGPROBS_SAMPLING（batch 级随机种子，topk 的 Exponential 取样）

参数	维度	数据类型	格式	描述
probs	[batch, voc_size]	float16/bf16	ND	输入，词表概率。
topk	[batch, 1]	int32	ND	输入top-k，top-k截取的位置，batch的值需与probs的一致。
topp	[batch, 1]	float16/bf16	ND	输入top-p，top-p截取的概率，batch的值需与probs的一致。
exp	[batch, voc_size]	float16/bf16	ND	输入，所除的指数分布，维度需与probs的一致。
sampled_indices	[batch, 1]	int32	ND	输出，取样的序号。
sampled_probs	[batch, 1]	float16/bf16	ND	输出，取样的值。
sampled_logprobs	[batch, logProbsSize]	float	ND	输出，topkToppSamplingType = BATCH_TOPK_EXPONENTIAL_LOGPROBS_SAMPLING时有效。值为对应概率值的对数。

 版权所有 © 华为技术有限公司

10.67.5 规格约束

- probs必须是两维张量。
- batch需小于等于512。
- logProbsSize需要小于等于voc_size。

10.67.6 功能列表

10.67.6.1 非 batch 级随机种子、topk 取样

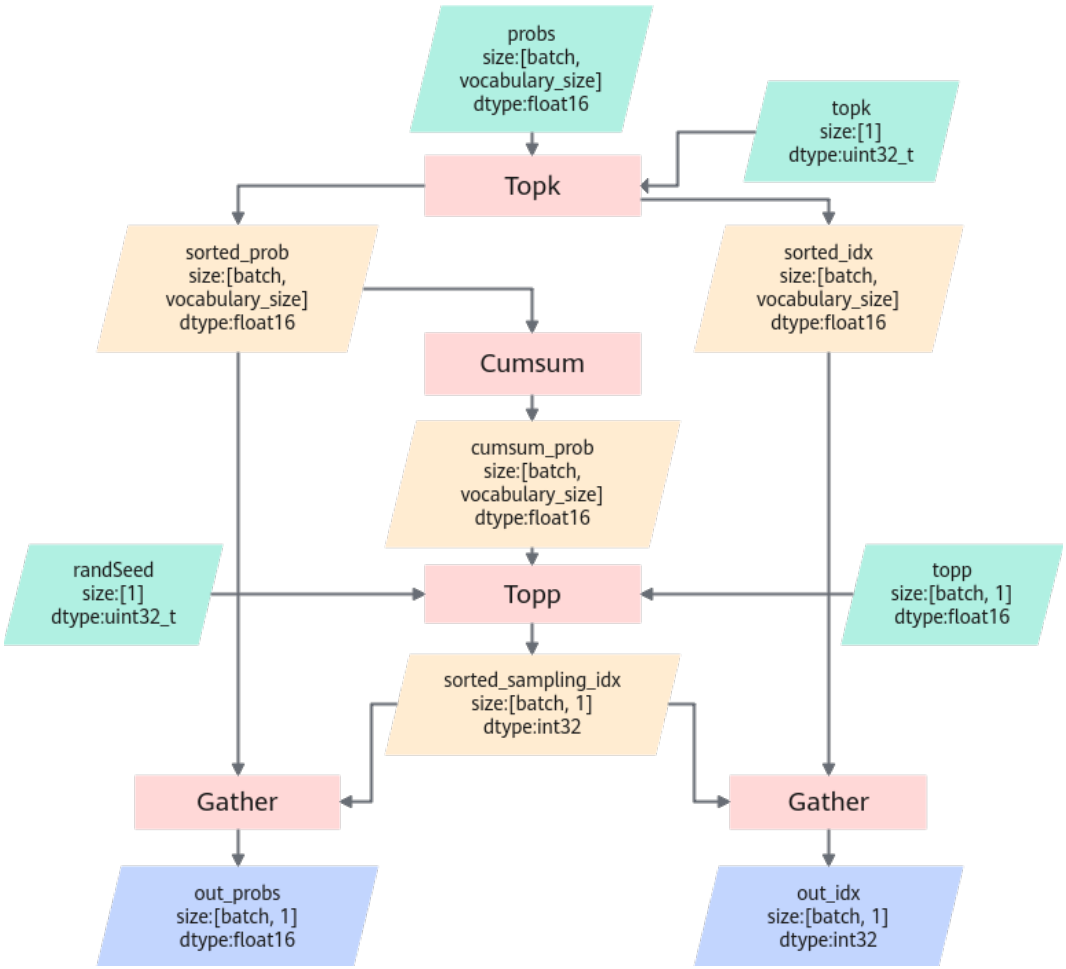
使用单一的randSeed和topk，对输入的词表概率的所有batch使用统一的topk值进行截取，再对截取后的词表概率的所有batch使用统一的randSeed进行topp取样。

使用的参数

topkToppSamplingType = SINGLE_TOPK_SAMPLING、topk、randSeed

流程图

图 10-76 非 batch 级随机种子 topk 取样流程图



 版权所有 © 华为技术有限公司

10.67.6.2 batch 级别随机种子、topk 的 multinomial 取样

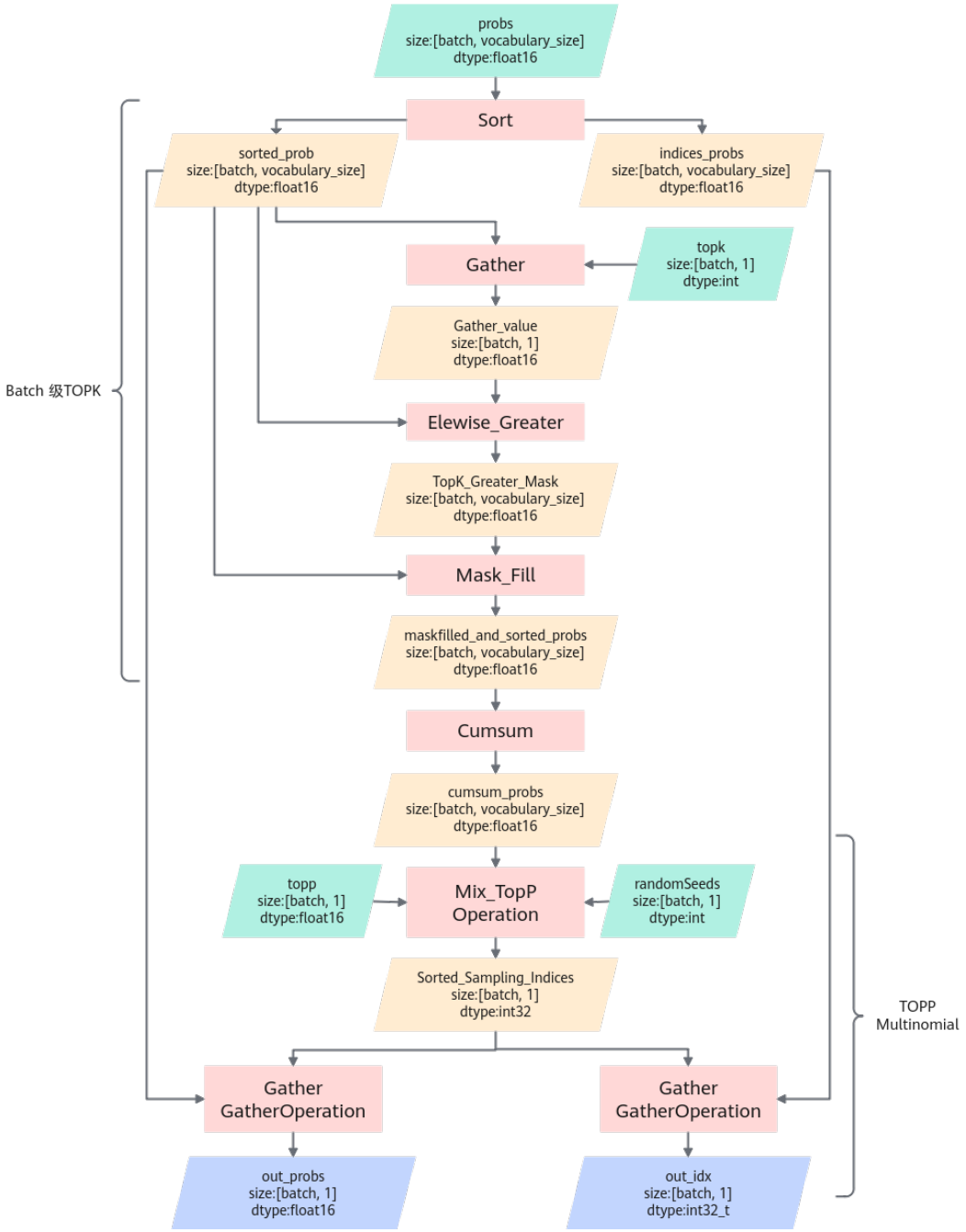
对于词表概率的每一个batch使用不同的topk进行截取，再对截取后的词表概率的每个batch使用不同的随机种子进行topp的multinomial取样。

使用的参数

topkToppSamplingType = BATCH_TOPK_MULTINOMIAL_SAMPLING、randSeeds

流程图

图 10-77 batch 级别随机种子 topk 的 multinomial 取样流程图



 版权所有 © 华为技术有限公司

10.67.6.3 batch 级随机种子，topk 的 Exponential 取样

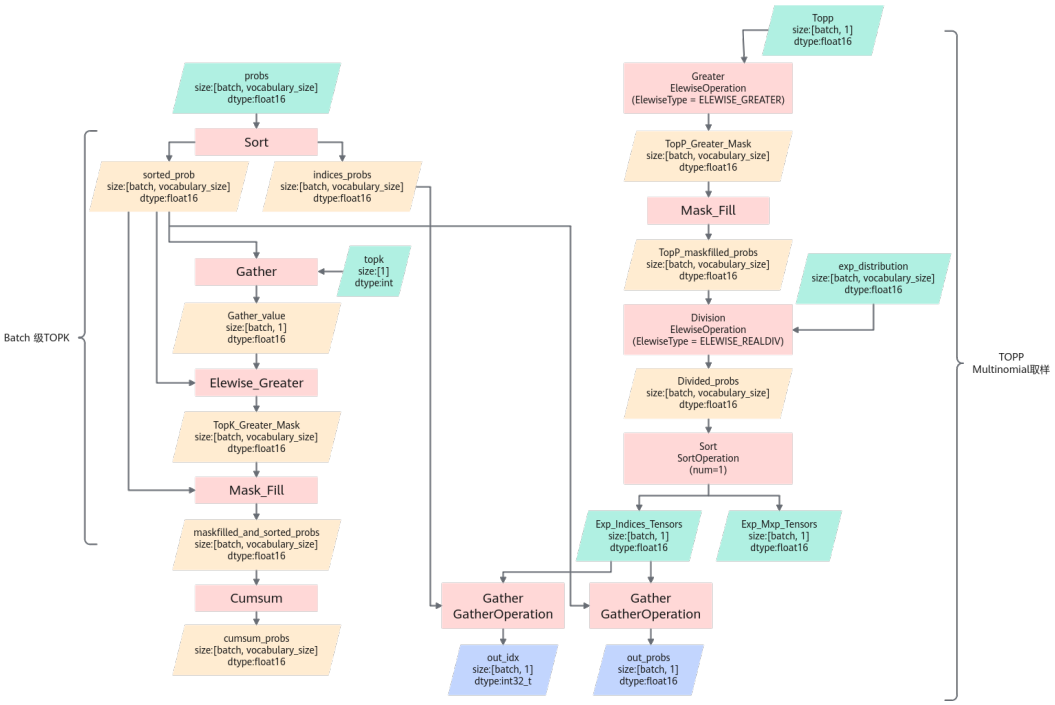
对于词表概率的每一个batch使用不同的topk进行截取，再对截取后的词表概率的每个batch使用不同的随机种子进行topp的exponential取样。

使用的参数

topkToppSamplingType = BATCH_TOPK_EXPONENTIAL_SAMPLING

流程图

图 10-78 batch 级随机种子 topk 的 Exponential 取样流程图



10.67.6.4 batch 级别随机种子、topk 的 logprobs multinomial 取样

对于词表概率的每一个batch使用不同的topk进行截取，再对截取后的词表概率的每个batch使用不同的随机种子进行topp的multinomial取样。

使用的参数

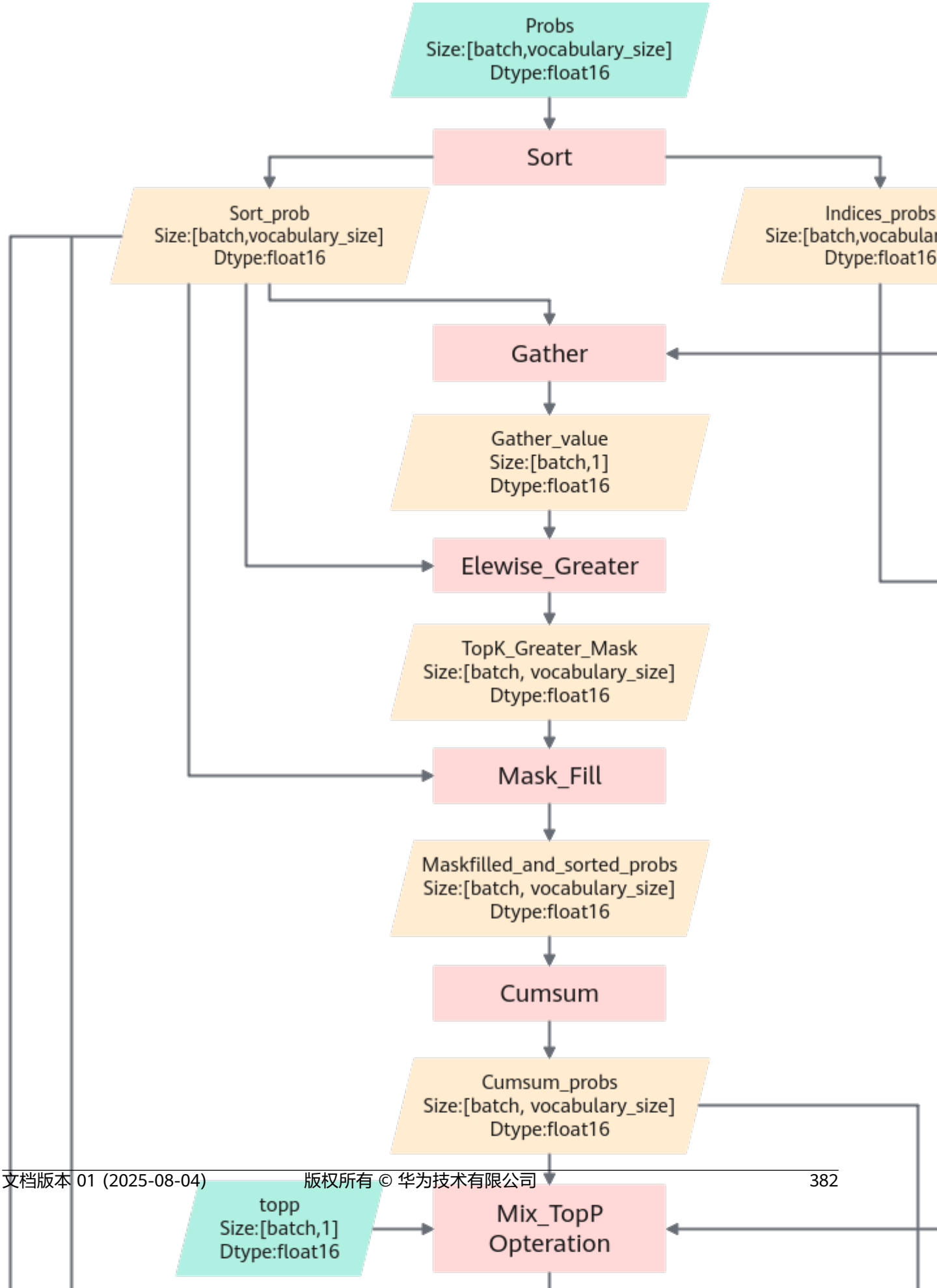
topkToppSamplingType = BATCH_TOPK_MULTINOMIAL_LOGPROBS_SAMPLING、logProbsSize

 版权所有 © 华为技术有限公司

流程图

 版权所有 © 华为技术有限公司

图 10-79 batch 级别随机种子 topk 的 logprobs multinomial 取样流程图



 版权所有 © 华为技术有限公司

10.67.6.5 batch 级随机种子，topk 的 logprobs Exponential 取样

对于词表概率的每一个batch使用不同的topk进行截取，再对截取后的词表概率的每个batch使用不同的随机种子进行topp的exponential取样。

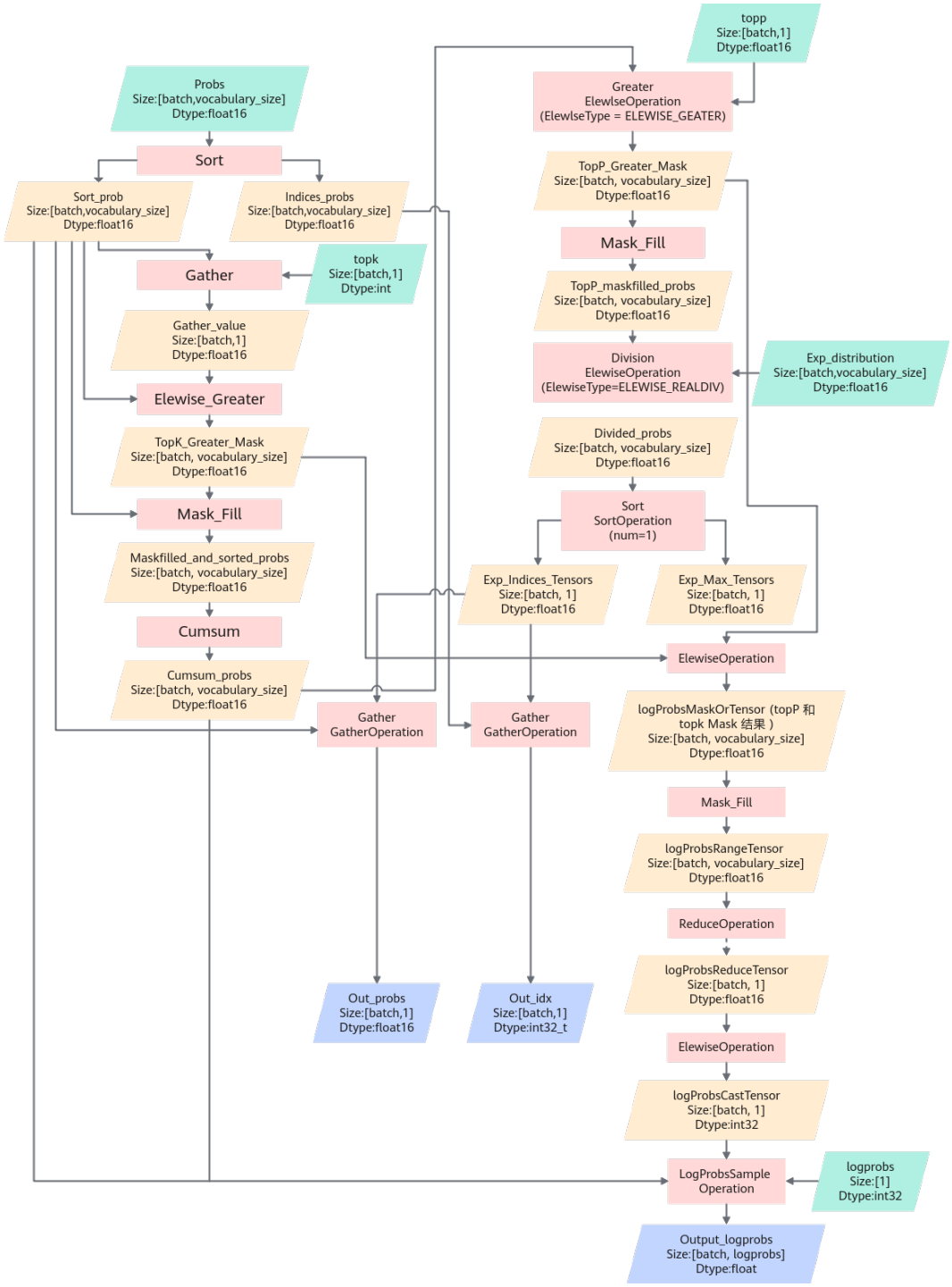
使用的参数

topkToppSamplingType = BATCH_TOPK_EXPONENTIAL_LOGPROBS_SAMPLING、logProbsSize

 版权所有 © 华为技术有限公司

流程图

图 10-80 batch 级随机种子 topk 的 logprobs Exponential 取样流程图



 版权所有 © 华为技术有限公司

10.68 TransdataOperation

功能

数据格式转换处理。将ACL_FORMAT_ND格式转换为ACL_FORMAT_FRACTAL_NZ格式，或将ACL_FORMAT_FRACTAL_NZ格式转换为ACL_FORMAT_ND格式。

说明

- 等效描述：ND、ACL_FORMAT_ND
- 等效描述：NZ、ACL_FORMAT_FRACTAL_NZ、FRACTAL_NZ

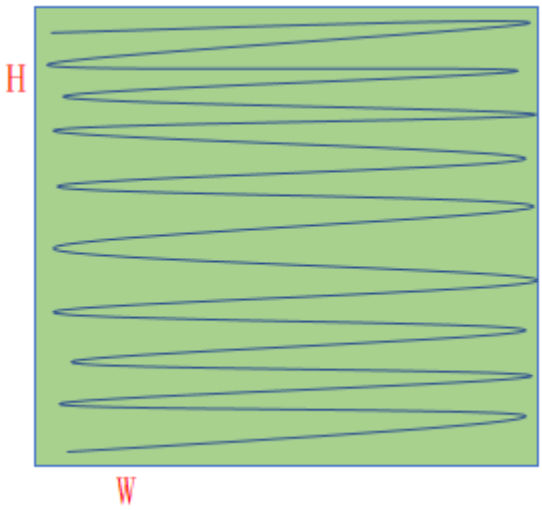
- **ACL_FORMAT_ND格式**
  - **aclFormat中的描述**

    ACL_FORMAT_ND = 2：表示支持任意格式，仅Square、Tanh等单输入算子适用。
  - **详细说明**

    Format为数据的物理排布格式，定义了解读数据的维度，ND格式表示1D、2D、3D、4D等。以4D为例，N：Batch数量、H：Height，特征图高度、W: Width, 特征图宽度、C: Channel，特征图通道。

    以2D为例，一般按照行优先的方式存储。

    图 10-81 ACL_FORMAT_ND 格式示例图

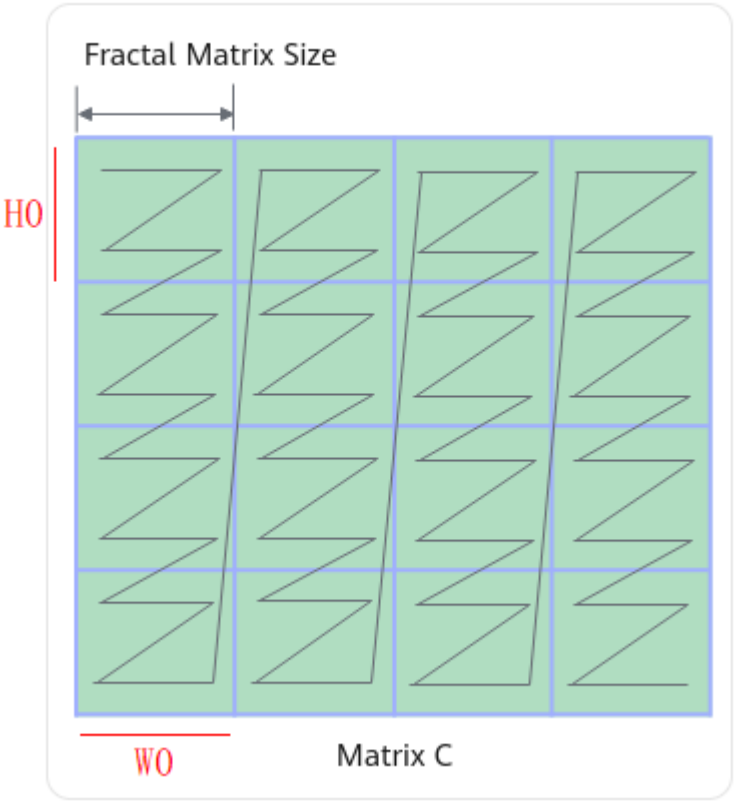
- **ACL_FORMAT_FRACTAL_NZ格式**
  - **aclFormat中的描述**

    ACL_FORMAT_FRACTAL_NZ = 29：内部格式，用户目前无需使用。
  - **详细说明**

    FRACTAL_NZ是分形格式（NW1H1H0W0），将整个矩阵分为(H1*W1)个分形，每一个分形内部有(H0*W0)个元素对于每一个N层。

 版权所有 © 华为技术有限公司

图 10-82 ACL_FORMAT_FRACTAL_NZ 格式示例图



以上图为例，N=1、W1=4、H1=4。

硬件支持情况

硬件型号	支持情况	特殊说明
Atlas 200I/500 A2 推理产品	支持	仅支持float16数据类型。
Atlas 推理系列产品	支持	-
Atlas 800I A2 推理产品/Atlas A2 训练系列产品	支持	-

定义

```
struct TransdataParam {
    enum TransdataType : int {
        UNDEFINED = 0,
        FRACTAL_NZ_TO_ND,
        ND_TO_FRACTAL_NZ
    };
    TransdataType transdataType = UNDEFINED;
    SVector<int64_t> outCrops = {0, 0};
    uint8_t rsv[8] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	描述
transdataType	TransdataType	UNDEFINED	数据格式转换类型，支持FRACTAL_NZ和ND互相转换。 ● UNDEFINED：默认。 ● FRACTAL_NZ_TO_ND：FRACTAL_NZ转ND。 ● ND_TO_FRACTAL_NZ：ND转FRACTAL_NZ。 此默认类型不可用，用户需配置此项参数。
outCrops	SVector<int64_t>	{0,0}	仅当FRACTAL_NZ转ND时使用，表示原ND数据格式Shape的最后两维。
rsv[8]	uint8_t	{0}	预留参数。

ND 转 NZ 输入输出描述

参数	维度	数据类型	格式	描述
x	1. [batch, m, n] 2. [m, n]	float16/int8/bf16	ND	输入tensor。
y	[batch, n1, m1m0, n0]	float16/int8/bf16	NZ	输出tensor，数据类型与输入tensor一致。 ● 当输入tensor数据类型为float16/bf16时，n0, m0为16。 ● 当输入tensor数据类型为int8时，n0为32，m0为16。 ● 当输入tensor为2维时，输出tensor的batch为1。

NZ 转 ND 输入输出描述

参数	维度	数据类型	格式	描述
x	[batch, n1, m1m0, n0]	float16/bf16	NZ	输入tensor。
y	[batch, m, n]	float16/bf16	ND	输出tensor，数据类型与输入tensor一致。

 版权所有 © 华为技术有限公司

功能列表

- **将ACL_FORMAT_ND数据格式的Tensor转换为ACL_FORMAT_FRACTAL_NZ数据格式**
  - **数据类型为float16**
    - 输入tensor维度为2

      将一个shape为[m,n]的Tensor作为输入，m0 = n0 = 16。输出tensor的shape为[1,⌈n/n0⌉,⌈m/m0⌉∗m0,n0]。
    - 输入tensor维度为3

      将一个shape为[batch,m,n]的Tensor作为输入，m0 = n0 = 16。输出tensor的shape为[batch,⌈n/n0⌉,⌈m/m0⌉∗m0,n0]。
  - **数据类型为int8**
    - 输入tensor维度为2

      将一个shape为[m,n]的Tensor作为输入，m0 = 16，n0 = 32。输出tensor的shape为[1,⌈n/n0⌉,⌈m/m0⌉∗m0,n0]。
    - 输入tensor维度为3

      将一个shape为[batch,m,n]的Tensor作为输入，m0 = 16，n0 = 32。输出tensor的shape为[batch,⌈n/n0⌉,⌈m/m0⌉∗m0,n0]。
- **将ACL_FORMAT_FRACTAL_NZ数据格式的Tensor转换为ACL_FORMAT_ND数据格式**
  - **数据类型为float16**

    将一个shape为[b, n1, m1m0, n0]的Tensor作为输入，参数中的outCrops = {m, n}。输出tensor的shape为[b, m, n]，其中m的取值范围为 $\left(\left\lfloor\frac{m1m0}{16}\right\rfloor*16, \left\lceil\frac{m1m0}{16}\right\rceil*16\right]$，n的取值范围为 $\left(\left\lfloor\frac{n1n0}{16}\right\rfloor*16, \left\lceil\frac{n1n0}{16}\right\rceil*16\right]$。

规格约束

- 使用的NZ的dims约定表示方式：{b, n1, m1m0, n0}，对应的ND的dims是{b, m, n}，其中：b表示batch，如果batch为1，该维度为1，不可省略。如果batch有多个，该维度为所有batch维度合轴的结果。

  m0/n0表示对齐位，float16时，n0与m0都为16, int8时，n0为32，m0为16，m1m0表示原始ND的m维度经过对齐位向上对齐，n1表示原始ND的n维度经过对齐位向上对齐后，除以n0的商。例如原始ND的dims为{8, 100, 30}，则其对应的NZ的dims为{8, 2, 112, 16}。
- outCrops的长度要求是2，其值须满足以下要求：
  - 如果m0m1落在区间$(k_1 \times 16, (k_1 + 1) \times 16]$（其中$k_1$为正整数）内，那么该区间即为outCrops[0]的取值范围要求。
  - 如果n0*n1落在区间$(k_2 \times 16, (k_2 + 1) \times 16]$（其中$k_2$为正整数）内，那么该区间即为outCrops[1]的取值范围要求。
- 不支持原地写。

 版权所有 © 华为技术有限公司

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include <random>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const uint32_t BATCH_SIZE = 8;    // 批处理大小
const uint32_t SEQ_LEN = 100;     // 序列长度
const uint32_t HIDDEN_SIZE = 30;  // 隐藏层维度

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> 返回一个{[BATCH_SIZE, SEQ_LEN, HIDDEN_SIZE]}的输入Tensors
 */
atb::SVector<atb::Tensor> PrepareInTensors(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建一个[BATCH_SIZE*SEQ_LEN*HIDDEN_SIZE]的vector，其中各个值为取值范围为[-100,100)的随机数
    std::vector<float> inTensorData(BATCH_SIZE * SEQ_LEN * HIDDEN_SIZE, 1.0);
    // 创建输入Tensor
    atb::Tensor inTensor = CreateTensorFromVector(
        contextPtr, stream, inTensorData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {BATCH_SIZE,
SEQ_LEN, HIDDEN_SIZE});
    atb::SVector<atb::Tensor> inTensors = {inTensor};
    return inTensors;
}

/**
 * @brief 创建一个 ND转NZ的TransdataOperation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::TransdataParam opParam;
    opParam.transdataType = atb::infer::TransdataParam::TransdataType::ND_TO_FRACTAL_NZ;
    atb::Operation *transdataOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(opParam, &transdataOp));
    return transdataOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // TransdataOp ND to NZ示例
    atb::Operation *transdataOp = PrepareOperation();
    // 准备VariantPack
    uint32_t ALIGN_16 = 16;
    atb::VariantPack variantPack;
```

 版权所有 © 华为技术有限公司

```
    variantPack.inTensors = PrepareInTensors(context, stream);  // 放入输入tensor
    atb::Tensor outTensor = CreateTensor(ACL_FLOAT16,
        aclFormat::ACL_FORMAT_FRACTAL_NZ,
        {BATCH_SIZE,
            (HIDDEN_SIZE + ALIGN_16 - 1) / ALIGN_16,
            (SEQ_LEN + ALIGN_16 - 1) / ALIGN_16 * ALIGN_16,
            ALIGN_16});
    variantPack.outTensors.push_back(outTensor);  // 放入输出tensor
    uint64_t workspaceSize = 0;
    // Transdata ND to NZ准备工作
    CHECK_STATUS(transdataOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // Transdata ND to NZ执行
    transdataOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(transdataOp));  // operation，对象概念，先释放
    std::cout << "Transdata ND to NZ demo success!" << std::endl;

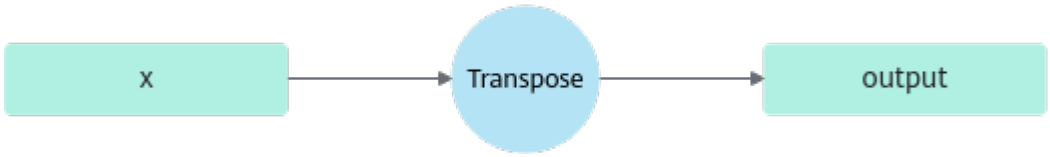
    // 资源释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(atb::DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    return 0;
}
```

10.69 TransposeOperation

功能

改变输入Tensor的排列顺序，在多个维度上进行转置。

图 10-83 TransposeOperation



定义

```
struct TransposeParam {
    SVector<int32_t> perm;
    uint8_t rsv[8] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	描述
perm	SVector<int32_t>	-	指示输入维度的重排结果。 例如：输入tensor的shape是[a,b,c]，perm是[0,2,1]，表明交换第一维和第二维，输出tensor的shape是[a,c,b]。 perm需要保证输入正确，维度和输入x一致。
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式
x	[dim_0, dim_1, ..., dim_n]	float16/bf16/int64/int8/int32	ND

输出

参数	维度	数据类型	格式
output	维度由参数确定	float16/bf16/int64/int8/int32 数据类型与x保持一致	ND

规格约束

Atlas 200I/500 A2 推理产品上仅支持float16数据类型。

算子调用示例（C++）

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
#include "atb/types.h"
#include "atb/atb_infer.h"

#include "demo_util.h"

const uint32_t DIM1 = 2;
const uint32_t DIM2 = 3;
```

 版权所有 © 华为技术有限公司

```
/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建x tensor
    std::vector<float> xData(DIM1 * DIM2, 1.0);
    std::vector<int64_t> xShape = {DIM1, DIM2};
    atb::Tensor tensorX =
        CreateTensorFromVector(contextPtr, stream, xData, ACL_FLOAT16, aclFormat::ACL_FORMAT_ND,
xShape);
    atb::SVector<atb::Tensor> inTensors = {tensorX};
    return inTensors;
}

/**
 * @brief 创建一个Reduce的Operation，并设置参数
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *PrepareOperation()
{
    atb::infer::TransposeParam transposeOpParam;
    atb::SVector<int32_t> perm = {1, 0};
    transposeOpParam.perm = perm;
    atb::Operation *transposeOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(transposeOpParam, &transposeOp));
    return transposeOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    CHECK_STATUS(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_STATUS(aclrtSetDevice(deviceId));
    atb::Context *context = nullptr;
    CHECK_STATUS(atb::CreateContext(&context));
    void *stream = nullptr;
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // Transpose示例
    atb::Operation *transposeOp = PrepareOperation();
    // 准备输入tensor
    atb::VariantPack transposeVariantPack;
    transposeVariantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    atb::Tensor tensorOut = CreateTensor(ACL_FLOAT16, aclFormat::ACL_FORMAT_ND, {DIM2, DIM1});
    transposeVariantPack.outTensors = {tensorOut};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspace大小
    CHECK_STATUS(transposeOp->Setup(transposeVariantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // reduce执行
    transposeOp->Execute(transposeVariantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成
    for (atb::Tensor &inTensor : transposeVariantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
```

 版权所有 © 华为技术有限公司

```
    }
    CHECK_STATUS(atb::DestroyOperation(transposeOp)); // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context)); // context，全局资源，后释放
    CHECK_STATUS(aclrtResetDevice());
    CHECK_STATUS(aclFinalize());
    std::cout << "Transpose demo success!" << std::endl;
    return 0;
}
```

10.70 UnpadOperation

功能

对于输入input_ids，把所有有效的token拼接在一起，并在最后补0。

使用场景

为了保证[batch,max_seq_len]的二维shape,某些batch的真实句子长度会被扩展成max_seq_len的长度，此算子目的为去除这些padding的部分。

使用示例：

假设有以下输入：

- input_ids：[[1, 2, 3, 0, 0], [4, 5, 0, 0, 0]]
- cum_offsets_now：[[2], [5]]
- seq_len：[[3], [2]]

在这个例子中：

- 第一个 batch 有 2 个填充 0，因此 cum_offsets_now 的第一个值是 2。
- 第二个 batch 有 5 个填充 0（包括第一个 batch 的 2 个填充 0），因此 cum_offsets_now 的第二个值是 5。

通过 unpad 算子处理后：

1. **计算 cum_offsets_out：**
   去除最后一个 batch 的数值：[[2]]
   整体后移一个 batch：[[0], [2]]
   第一个 batch 的值置为 0：[[0], [2]]（cum_offsets_out的值）
2. **移除填充 0：**
   - 对于第一个 batch：
     - cum_offsets_out[0] 是 0，表示从位置 0 开始。
     - seq_len[0] 是 3，表示有 3 个有效 token。
     - 将 input_ids[0] 中的前 3 个有效 token 复制到 x_remove_padding 的前 3 个位置。
   - 对于第二个 batch：
     - cum_offsets_out[1] 是 2，表示从位置 2 开始。

 版权所有 © 华为技术有限公司

- seq_len[1] 是 2，表示有 2 个有效 token。
- 将 input_ids[1] 中的前 2 个有效 token 复制到 x_remove_padding 的第 3 和第 4 个位置。
- 根据 cum_offsets_out 和 seq_len，重新排列有效 token，得到 x_remove_padding：[1, 2, 3, 4, 5, 0, 0, 0, 0, 0]。

应用场景：

在自然语言处理（NLP）中，输入的句子长度通常不同。为了让它们在一个 batch 中处理，我们需要将它们填充（pad）到相同的长度。填充的方式通常是在句子的末尾添加零（0），直到所有句子长度相同。Unpad 用于在处理完这些填充的序列后，去除填充的 0，提取有效的 token 序列。

定义

```
struct UnpadParam {
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
input_ids	[batch, max_seq_len]	int64	ND	输入tensor：经过pad（末尾填充0）之后，batch个token id序列。
cum_offsets_now	[batch, 1]	int32	ND	输入tensor：每个batch末尾填充0的数量组成的序列的前缀和。
token_num	[batch, 1]	int64	ND	输入tensor：有效token数量总和。
seq_len	[batch, 1]	int32	ND	输入tensor：每个batch中的有效token数量。

 版权所有 © 华为技术有限公司

输出

参数	维度	数据类型	格式	描述
x_remove_padding	[1, batch * max_seq_len]	int64	ND	输出tensor：从input_ids中去除填充的0后得到的有效token序列（为了保持shape，在末尾填充0）。
cum_offsets_out	[batch, 1]	int32	ND	输出tensor：去除cum_offsets_now最后一个batch的数值，再将其整体后移一个batch，并将第一个batch的值置为0。
padding_offset	[1, batch * max_seq_len]	int32	ND	输出tensor：由seq_len和cum_offsets_out组合计算得到。

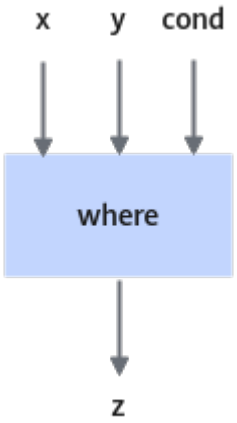
10.71 WhereOperation

功能

三目运算。

输入张量为cond, x, y，输出张量 z = cond ? x : y。

算子上下文



算子功能实现

- **输入**：
  cond（条件张量）：定义了每个元素的选择条件。
  x：当条件为 1时，选择的第一个输入张量。
  y：当条件为0时，选择的第二个输入张量。

 版权所有 © 华为技术有限公司

- **输出**：输出张量的每个元素根据对应位置的条件从x或y中选取，返回一个和x、y形状相同的张量。 z = (cond == 1 ? x : y)

定义

```
struct WhereParam {
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
cond	[dim_0, dim_1, ..., dim_n]	int8	ND	输出tensor1，条件变量。
x	[x_dim_0, x_dim_1, ..., x_dim_n]	float16	ND	输入tensor2。
y	[y_dim_0, y_dim_1, ..., y_dim_n]	float16	ND	输入tensor3。

输出

参数	维度	数据类型	格式	描述
z	[z_dim_0, z_dim_1, ..., z_dim_n]	float16	ND	输出tensor

规格约束

输入cond的元素只能是0或者1。输出z的维度为输入x与y广播后的结果。要求cond, x, y必须是可广播的。

接口调用示例

输入：

```
cond = [[1, 0],
        [0, 1]]
x = [[1, 2],
     [3, 4]]
y = [[10, 20],
     [30, 40]]
```

输出：

 版权所有 © 华为技术有限公司

```
z = [[ 1, 20],
     [30, 4]]
```

 版权所有 © 华为技术有限公司

11 atb/train_op_params.h

operator==
FastSoftMaxOperation
FastSoftMaxGradOperation
GenAttentionMaskOperation
LaserAttentionOperation
LaserAttentionGradOperation
PadWithHiddenStateOperation
RmsNormBackwardOperation
RopeGradOperation
StridedBatchMatmulOperation
UnpadWithHiddenStateOperation

11.1 operator==

功能

判断参数是否相同。

定义

```
inline bool operator==(const GenAttentionMaskParam &left, const GenAttentionMaskParam &right)
{
    return left.headNum == right.headNum && left.seqLen == right.seqLen;
}
inline bool operator==(const RopeGradParam &left, const RopeGradParam &right)
{
    return left.qSeqLen == right.qSeqLen;
}
inline bool operator==(const FastSoftMaxParam &left, const FastSoftMaxParam &right)
{
    return left.headNum == right.headNum && left.qSeqLen == right.qSeqLen;
}
inline bool operator==(const FastSoftMaxGradParam &left, const FastSoftMaxGradParam &right)
```

 版权所有 © 华为技术有限公司

```
{
    return left.headNum == right.headNum && left.qSeqLen == right.qSeqLen;
}
inline bool operator==(const StridedBatchMatmulParam &left, const StridedBatchMatmulParam &right)
{
    return left.transposeA == right.transposeA && left.transposeB == right.transposeB && left.batch ==
right.batch &&
        left.headNum == right.headNum && left.m == right.m && left.n == right.n && left.k == right.k &&
        left.lda == right.lda && left.ldb == right.ldb && left.ldc == right.ldc && left.strideA == right.strideA
&&
        left.strideB == right.strideB && left.strideC == right.strideC;
}
inline bool operator==(const UnpadWithHiddenStateParam &left, const UnpadWithHiddenStateParam
&right)
{
    return left.qSeqLen == right.qSeqLen && left.maxSeqLen == right.maxSeqLen;
}
inline bool operator==(const PadWithHiddenStateParam &left, const PadWithHiddenStateParam &right)
{
    return left.qSeqLen == right.qSeqLen && left.maxSeqLen == right.maxSeqLen;
}
```

11.2 FastSoftMaxOperation

功能描述

训练算子，与unpadOperation一起使用，将unpad处理后的Q矩阵和K矩阵相乘的结果做高性能的Softmax处理。

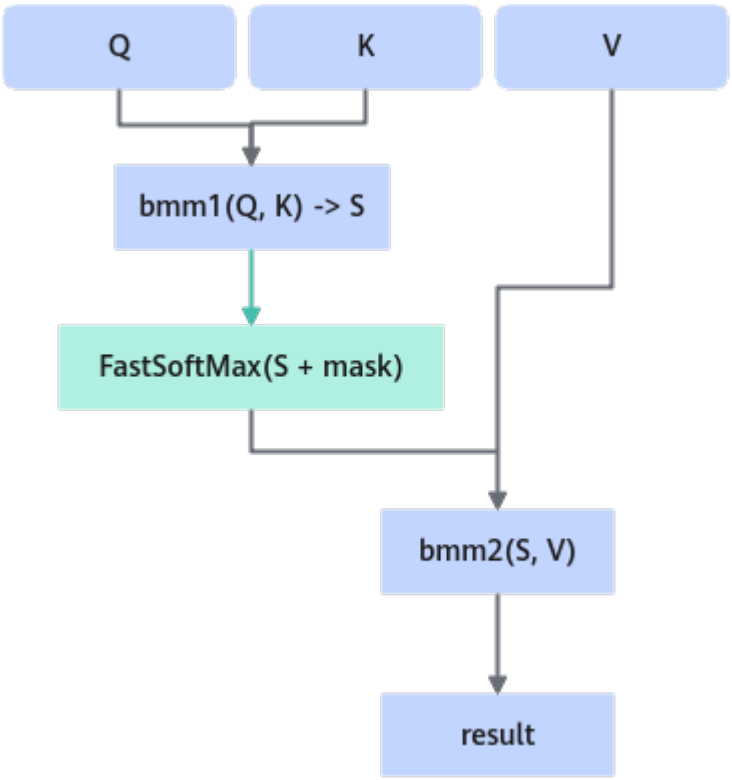
softmax数学公式：

$$\sigma\left(x \right)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{zk}} \; for\, j = 1, \ldots, K$$

 版权所有 © 华为技术有限公司

使用场景

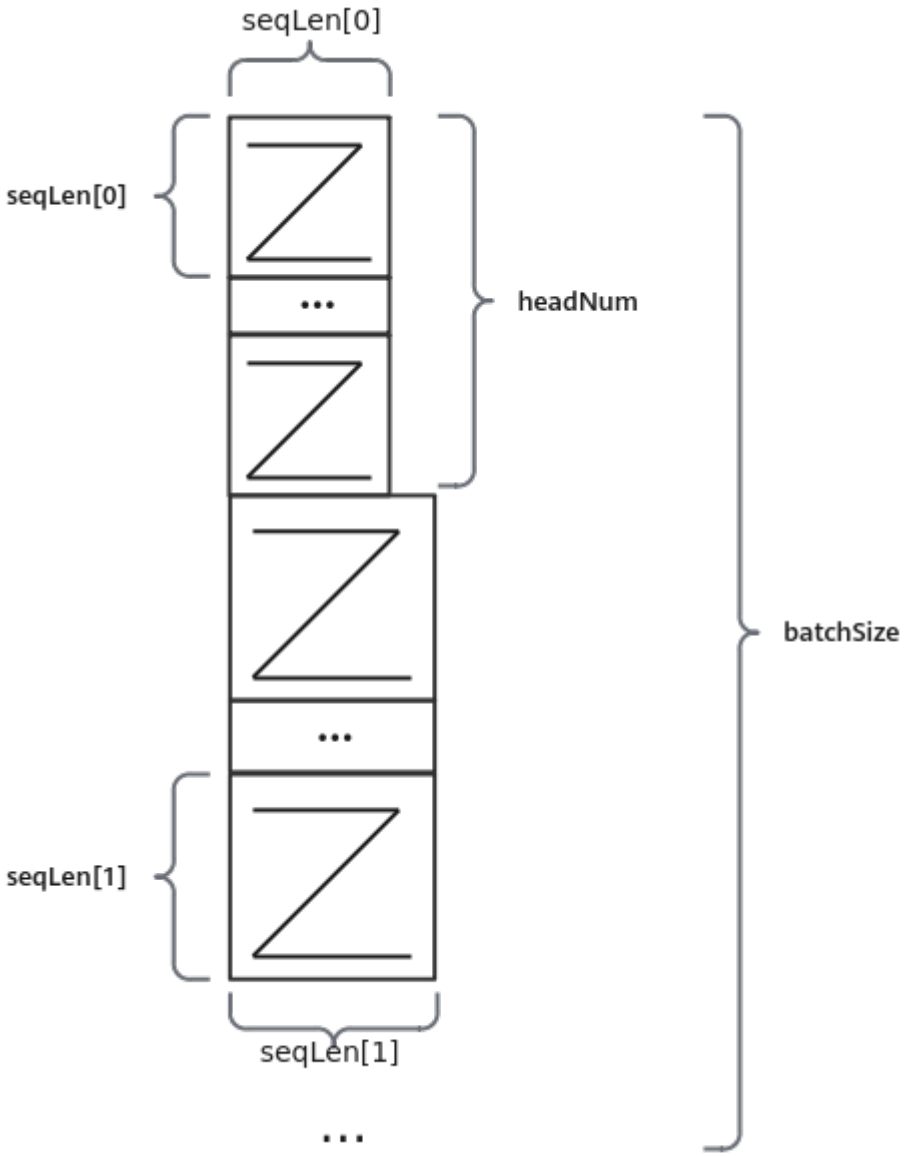
图 11-1 模型流程图



Multi-Head Attention计算过程如图11-1所示。经过unpad处理后，Q矩阵和K矩阵相乘的结果实际上由batchSize个大小为(seqLen[i], seqLen[i])的矩阵拼接而成（如图11-2所示），需要对这个结果的末轴做SoftMax操作。采用FastSoftMax算子，接收seqLen和headNum的信息可以对这种排列方式的数据实现SoftMax操作。

 版权所有 © 华为技术有限公司

图 11-2 输入数据的排列方式



定义

```
struct FastSoftMaxParam {
    int32_t headNum = 0;
    std::vector<int32_t> qSeqLen;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
headNum	int32_t	0	Attention的head数量，需大于0。
qSeqLen	std::vector<int32_t>	-	每个batch的实际输入长度。元素个数为batchSize，最大不超过32。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
inTensor	[nSquareTokens]	float16	ND	输入tensor，是batch个(headNum, qSeqLen[i], qSeqLen[i])大小的矩阵按照ND排布拼接成一维的结果。

输出

参数	维度	数据类型	格式	描述
outTensor	[nSquareTokens]	float16	ND	输出tensor，数据范围在[0, 1]之间。

规格约束

- qSeqLen数组长度不超过32，且要求各元素大于0。
- 当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 输入tensor（inTensor）、输出tensor（outTensor）的维度大小nSquareTokens与参数中的headNum和qSeqLen有关。须满足

$$nSquareTokens = \sum_{k=0}^{n} (qSeqLen_k)^2 * headNum$$

$qSeqLen_k$：表示qSeqLen中的第k个元素。

功能列表

当参数列表中

$$qSeqLen = [qSeqLen_0, qSeqLen_1, ..., qSeqLen_n], headNum = head$$

时，

其对应的输入tensor的shape大小为

$$[nSquareTokens], nSquareTokens = \sum_{k=0}^{n} (qSeqLen_k)^2 * head$$

。

 版权所有 © 华为技术有限公司

例如参数中设置

$$qSeqLen=[487,16，532，413], headNum=8$$

此时的输入tensor的shape大小为

$$nSquareTokens=487^2*8+16^2*8+532^2*8+413^2*8=5528144$$

输入和输出tensor的shape为[5528144]。

11.3 FastSoftMaxGradOperation

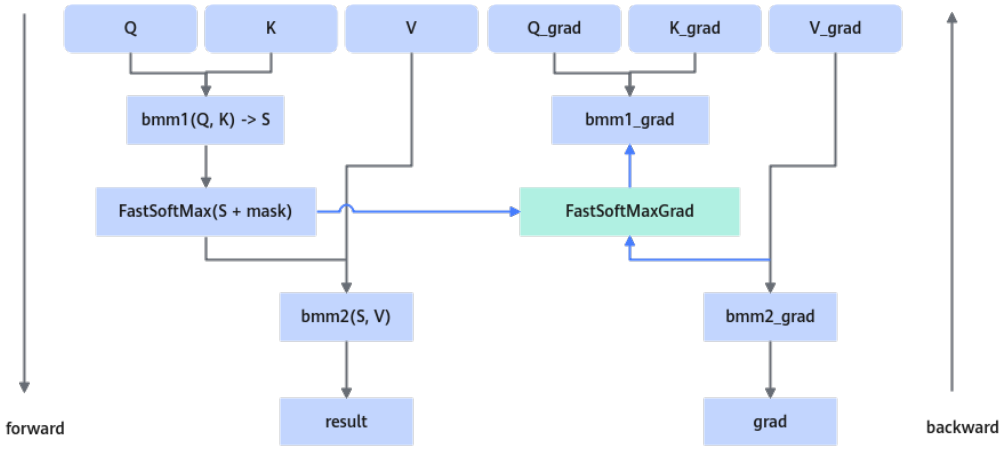
功能描述

训练算子，用于计算FastSoftMax的反向计算结果，其数据的排列结构与FastSoftMax相同。

$$nSquareTokens = headNum \sum_{i=0}^{batchSize-1} (qSeqLen[i])^2$$

使用场景

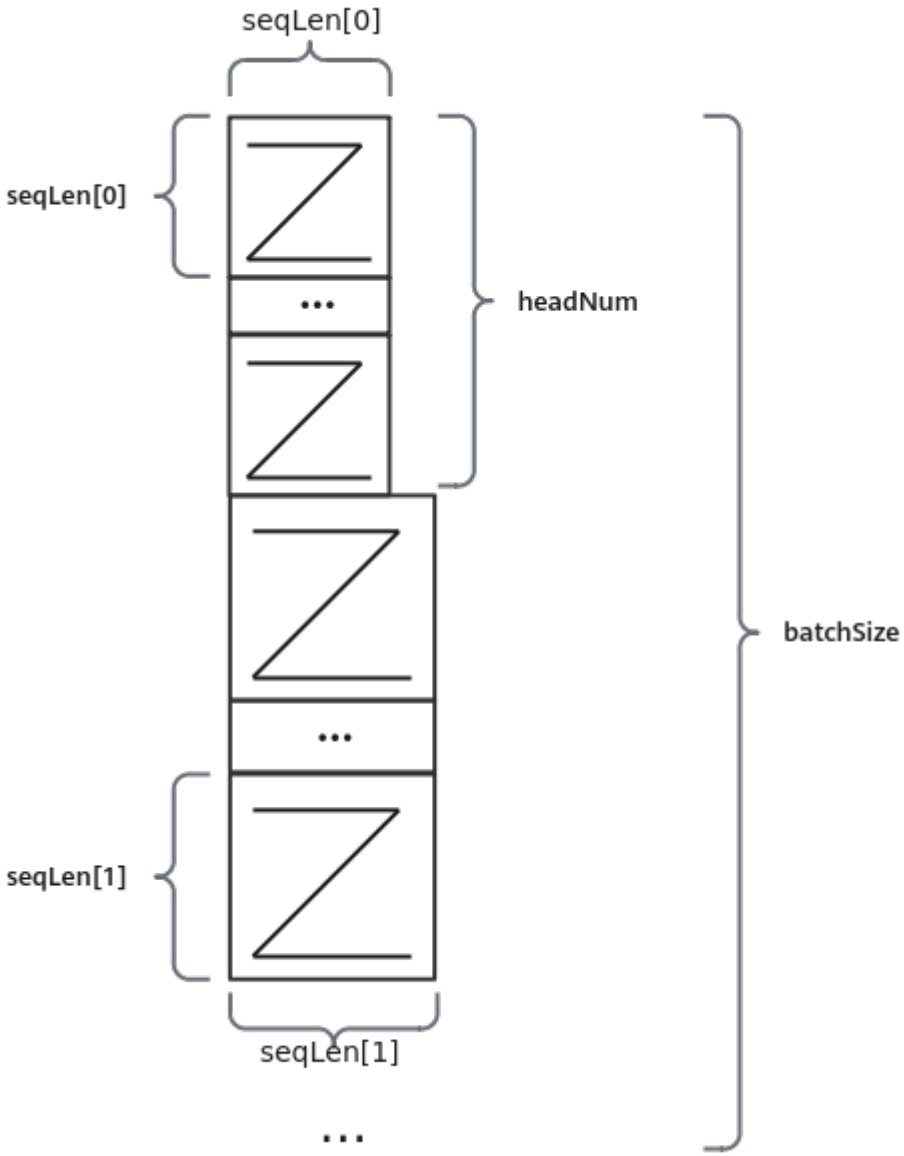
图 11-3 模型流程图



FastSoftMaxGrad算子用于计算FastSoftMax的方向计算结果，其数据的排列结构与FastSoftMax相同。

 版权所有 © 华为技术有限公司

图 11-4 输入数据的排列方式



定义

```
struct FastSoftMaxGradParam {
    int32_t headNum = 0;
    std::vector<int32_t> qSeqLen;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
headNum	int32_t	0	Attention的head数目，要求大于0。
qSeqLen	std::vector<int32_t>	-	每个batch的实际输入长度，元素个数与batch大小相同，最大不超过32。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
yInput	[nSquareTokens]	float16	ND	输入tensor，SoftMax算子前向计算的结果是batch个值域在(0,1)之间，且末轴和为1的(headNum, qSeqLen[i], qSeqLen[i])大小的矩阵，按照ND排布拼接成一维的结果。
yGrad	[nSquareTokens]	float16	ND	下一个算子传入的梯度数据。排列方式与yInput相同。

输出

参数	维度	数据类型	格式	描述
output	[nSquareTokens]	float16	ND	排列方式与yInput相同。

规格约束

- qSeqLen数组长度不超过32，且要求各元素大于0。
- 当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 输入tensor（yInput和yGrad）、输出tensor（outTensor）的维度大小nSquareTokens与参数中的headNum和qSeqLen有关。须满足

$$nSquareTokens = \sum_{k=0}^{n} (qSeqLen_k)^2 * headNum$$

$qSeqLen_k$：表示qSeqLen中的第k个元素。

功能列表

当参数列表中

$$qSeqlen = [qSeqLen_0, qSeqLen_1, ..., qSeqLen_n], headNum = head$$

时，

 版权所有 © 华为技术有限公司

其对应的yInput的shape大小为

$$\left[nSquareTokens \right], nSquareTokens = \sum_{k=0}^{n} (qSquare_k)^2 * head$$

。

例如参数中设置

$$qSeqLen = [487, 16, 532, 413], headNum = 8$$

此时的yInput的shape大小为

$$nSquareTokens = 487^2 * 8 + 16^2 * 8 + 532^2 * 8 + 413^2 * 8 = 5528144$$

输入yInput、yGrad和输出tensor的shape为[5528144]。

11.4 GenAttentionMaskOperation

功能

将attentionMask根据每个batch的实际seqlen进行转化，得到结果为一维tensor。

定义

```
struct GenAttentionMaskParam {
    int32_t headNum = 1;
    atb::SVector<int32_t> seqLen;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
headNum	int32_t	1	多头注意力机制的head数。
seqLen	atb::SVector<int32_t>	-	存储unpad场景下每个batch实际seqlen的值。元素个数为batchSize，最大为32。
rsv[8]	uint8_t	{0}	预留参数。

输入输出

参数	维度	数据类型	格式	描述
x	[batchSize, 1, maxSeqLen, maxseqlen]	float16	ND	输入，用于attentionmask计算的随机矩阵。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
output	[nSquareTokens]	float16	ND	输出，attentionmask计算的结果矩阵。

其中nSquareTokens的计算公式为：

$$nSquareTokens = headNum \sum_{i=0}^{batchSize-1} (qSeqLen[i])^2$$

规格约束

- 当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- qSeqLen数组长度不超过32，且要求各元素大于0。

11.5 LaserAttentionOperation

功能

训练场景下，使用LaserAttention算法实现self-attention（自注意力）的计算。

定义

```
struct LaserAttentionParam {
    int headNum = 0;
    std::string inputLayout = "BNSD";
    float scaleValue = 0.08838834764831843;
    float keepProb = 1.0f;
    int preTokens = 2147483647;
    int nextTokens = 1;
    int sparseMode = 0;
    int innerPrecise = 1;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	描述
headNum	int	0	(0, INT_MAX]	head个数，需大于0。
inputLayout	string	"BNSD"	"BNSD"	输入排布。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	描述
scaleValue	float	0.088388347648318443	(0, 1]	缩放系数。
keepProb	float	1.0f	目前暂未使用，配置值未生效，可配任意值	需要保留的比例，与dropMask相关。
preTokens	int	2147483647	[1, INT_MAX]，为256的整数倍	sparse纵向滑动窗口大小。与seqSize相关，约束参考**规格说明**。
nextTokens	int	1	目前暂未使用，配置值未生效，可配任意值	sparse横向滑动窗口大小。
sparseMode	int	0	目前暂未使用，配置值未生效，可配任意值	sparse模式，是否有下三角/梯形掩码。
innerPrecise	int	1	1	精度模式。1：高精度模式。
rsv[8]	uint8_t	{0}	-	预留参数。

输入

参数	维度	数据类型	格式	描述
query	[batch, qHeadNum, seqSize, q_head_size]	bf16	ND	注意力机制的Q矩阵。
key	[batch, kvHeadNum, kvSize, k_head_size]	bf16	ND	注意力机制的K矩阵。
value	[batch, kvHeadNum, kvSize, v_head_size	bf16	ND	注意力机制的V矩阵。
pseShift	-	-	-	目前为预留接口，输入空Tensor。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
dropMask	-	-	-	目前为预留接口，输入空Tensor。
paddingMask	-	-	-	目前为预留接口，输入空Tensor。
attenMask	[seqSize, kvSize]	bf16	ND	attention范围。 该接口可选，可正常输入，也可输入空Tensor不参与计算。
prefix	-	-	-	目前为预留接口，输入空Tensor。
actualSeqQLen	-	-	-	目前为预留接口，输入空Tensor。
actualSeqKVLen	-	-	-	目前为预留接口，输入空Tensor。

输出

参数	维度	数据类型	格式	描述
softmaxMax	[batch, qHeadNum, seqSize]	float	ND	输出tensor。
softmaxSum	[batch, qHeadNum, seqSize]	float	ND	输出tensor。
softmaxOut	[...]	bf16	ND	目前为预留接口，维度可任意配置。
attentionOut	[batch, qHeadNum, seqSize, v_head_size]	bf16	ND	输出tensor。

规格说明

- qHeadNum值与参数headNum值相同。
- qHeadNum值是kvHeadNum值的整数倍。
- seqSize值和kvSize值均为256的整数倍。
- 构造query/key/value时，若值域配置在[-100, 100]以内，采取uniform均匀分布方式生成数据；否则，采取normal正态分布，均值在[-100, 100]内随机选取，标准差在[1, 25]内随机选取。
- 当attenMask不为空Tensor时。
  - seqSize值和kvSize值相同。

 版权所有 © 华为技术有限公司

  - 为下三角形，当preTokens < seqSize时，算子内部当作梯形处理。
- q_head_size、k_head_size和v_head_size的值有以下组合：

q_head_size	k_head_size	v_head_size	约束说明
128	128	128	seqSize值不小于参数preTokens值。
192	256	128	seqSize值和参数preTokens值相同。

规格约束

- 当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 该算子涉及但不支持确定性计算。
- qHeadNum小于等于512。

11.6 LaserAttentionGradOperation

功能

训练场景下，使用LaserAttention算法实现self-attention（自注意力）的计算（反向）。

定义

```
struct LaserAttentionGradParam {
    int headNum = 0;
    std::string inputLayout = "BNSD";
    float scaleValue = 0.08838834764831843;
    float keepProb = 1.0f;
    int preTokens = 2147483647;
    int nextTokens = 1;
    int sparseMode = 0;
    int innerPrecise = 1;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	取值范围	描述
headNum	int	0	(0, INT_MAX]	head个数，需大于0。
inputLayout	string	"BNSD"	"BNSD"	输入排布。
scaleValue	float	0.08838834764831843	(0, 1]	缩放系数。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	取值范围	描述
keepProb	float	1.0f	目前暂未使用，配置值未生效，可配任意值	需要保留的比例，与dropMask相关。
preTokens	int	214748364747	[1, INT_MAX]，为256的整数倍	sparse纵向滑动窗口大小。与seqSize相关，约束参考**规格约束**。
nextTokens	int	1	目前暂未使用，配置值未生效，可配任意值	sparse横向滑动窗口大小。
sparseMode	int	0	目前暂未使用，配置值未生效，可配任意值	sparse模式，是否有下三角/梯形掩码。
innerPrecise	int	1	1	精度模式。1：高精度模式。
rsv[8]	uint8_t	{0}	-	预留参数。

输入

参数	维度	数据类型	格式	描述
query	[batch, qHeadNum, seqSize, q_head_size]	bf16	ND	注意力机制的Q矩阵。
key	[batch, kvHeadNum, kvSize, k_head_size]	bf16	ND	注意力机制的K矩阵。
value	[batch, kvHeadNum, kvSize, v_head_size]	bf16	ND	注意力机制的V矩阵。
attentionOutGrad	[batch, qHeadNum, seqSize, v_head_size]	bf16	ND	正向输出（attentionOut）的梯度。
pseShift	-	-	-	目前为预留接口，输入空Tensor。
dropMask	-	-	-	目前为预留接口，输入空Tensor。
paddingMask	-	-	-	目前为预留接口，输入空Tensor。

 版权所有 © 华为技术有限公司

参数	维度	数据类型	格式	描述
attenMask	[seqSize, kvSize]	float16	ND	attention范围。 该接口可选，可正常输入，也可输入空Tensor不参与计算。
softmaxMax	[batch, qHeadNum, seqSize]	float	ND	对应前向的输出softmaxMax。
softmaxSum	[batch, qHeadNum, seqSize]	float	ND	对应前向的输出softmaxSum。
softmaxIn	[...]	bf16	ND	对应前向的输出softmaxOut。维度可任意配置。
attentionIn	[batch, qHeadNum, seqSize, v_head_size]	bf16	ND	对应前向的输出attentionOut。
prefix	-	-	-	目前为预留接口，输入空Tensor。
actualSeqQLen	-	-	-	目前为预留接口，输入空Tensor。
actualSeqKVLen	-	-	-	目前为预留接口，输入空Tensor。

输出

参数	维度	数据类型	格式	描述
queryGrad	[batch, qHeadNum, seqSize, q_head_size]	bf16	ND	输出tensor。
keyGrad	[batch, kvHeadNum, kvSize, k_head_size]	bf16	ND	输出tensor。
valueGrad	[batch, kvHeadNum, kvSize, v_head_size]	bf16	ND	输出tensor。
dpse	[...]	bf16	ND	目前为预留接口，维度可任意配置。

规格说明

- qHeadNum值与参数headNum值相同。

- qHeadNum值是kvHeadNum值的整数倍。
- seqSize值和kvSize值均为256的整数倍。
- 构造query/key/value时，若值域配置在[-100, 100]以内，采取uniform均匀分布方式生成数据；否则，采取normal正态分布，均值在[-100, 100]内随机选取，标准差在[1, 25]内随机选取。
- 当attenMask不为空Tensor时：
  - seqSize值和kvSize值相同。
  - 为下三角形，当preTokens < seqSize时，算子内部当做梯形处理。
- attentionOutGrad值域为[-0.5, 0.5]。
- q_head_size、k_head_size和v_head_size的值有以下组合：

q_head_size	k_head_size	v_head_size	约束说明
128	128	128	seqSize值不小于参数preTokens值。
192	256	128	seqSize值和参数preTokens值相同。

规格约束

- 当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- 该算子涉及但不支持确定性计算。
- qHeadNum小于等于512。

11.7 PadWithHiddenStateOperation

功能

PadWithHiddenState是带隐藏层的pad，这个算子的主要作用是将变长的序列填充为固定长度，便于批处理和模型训练。

定义

```
struct PadWithHiddenStateParam {
    std::vector<int32_t> qSeqLen;
    int32_t  maxSeqLen = 4096;
    uint8_t rsv[12] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
qSeqLen	std::vector<int32_t>	-	每个batch的实际输入长度。元素个数为batchSize。batchSize的值最大不超过32。
maxSeqLen	int32_t	4096	qSeqLen中最大输入长度。取值不超过4096。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
rsv[12]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[nTokens, hiddenSize]	float16	ND	pad前不带冗余的tensor。与unpad输出shape一致。

- **计算公式**：

$$nTokens = \sum_{i=0}^{batchSize-1} qSeqLen[i]$$

这个公式用于计算一个batch中所有序列的总token数。

- **示例**：

batchSize = 3

qSeqlen = [5, 10, 7]

则 nTokens = 5 +10 + 7 =22

输出

参数	维度	数据类型	格式	描述
output	[batchSize, maxSeqLen, hiddenSize]	float16	ND	pad后带冗余的tensor。与unpad输入shape一致。

11.8 RmsNormBackwardOperation

功能

rmsnorm的反向计算。

定义

```
struct RmsNormBackwardParam {
    uint8_t rsv[8] = {0};
};
```

 版权所有 © 华为技术有限公司

参数列表

成员名称	类型	默认值	描述
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
dy	[dim_0, dim_1, ..., dim_n]	float16/ float/bf16	ND	输入梯度。数据类型、维度与x一致。
x	[dim_0, dim_1, ..., dim_n]	float16/ float/bf16	ND	正向计算输入。
rstd	[dim_0, dim_1, ..., dim_n]	float	ND	正向计算中间结果。
gamma	[dim_0, dim_1, ..., dim_n]	float16/ float/bf16	ND	数据类型与x一致。维度数需要大于0，并小于x的维度数，gamma的维度从最后一维向前，每一维都需要和x保持一致。

输出

参数	维度	数据类型	格式	描述
dx	[dim_0, dim_1, ..., dim_n]	float16/ float/bf16	ND	正向输入x的梯度。数据类型、维度与x一致。
dgamma	[dim_0, dim_1, ..., dim_n]	float	ND	正向输入gamma的梯度。维度与gamma一致。

规格约束

当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

算子调用示例

前置条件和编译命令请参见**4 算子调用示例**。

场景：基础场景。

```
#include <iostream>
#include <vector>
#include <numeric>
#include "acl/acl.h"
#include "atb/operation.h"
```

 版权所有 © 华为技术有限公司

```
#include "atb/types.h"
#include "atb/train_op_params.h"

#include "demo_util.h"

const int32_t DEVICE_ID = 0;
const uint32_t DIM_0 = 32;
const uint32_t DIM_1 = 64;
const uint32_t DIM_2 = 128;

/**
 * @brief 准备atb::VariantPack中的所有输入tensor
 * @param contextPtr context指针
 * @param stream stream
 * @return atb::SVector<atb::Tensor> atb::VariantPack中的输入tensor
 * @note 需要传入所有host侧tensor
 */
atb::SVector<atb::Tensor> PrepareInTensor(atb::Context *contextPtr, aclrtStream stream)
{
    // 创建shape为[32, 64, 128]的tensor
    atb::Tensor dy = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>(DIM_0 * DIM_1 * DIM_2, 2.0),
        ACL_FLOAT,
        aclFormat::ACL_FORMAT_ND,
        {DIM_0, DIM_1, DIM_2});
    atb::Tensor x = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>(DIM_0 * DIM_1 * DIM_2, 2.0),
        ACL_FLOAT,
        aclFormat::ACL_FORMAT_ND,
        {DIM_0, DIM_1, DIM_2});
    atb::Tensor rstd = CreateTensorFromVector(contextPtr,
        stream,
        std::vector<float>(DIM_0 * DIM_1, 2.0),
        ACL_FLOAT,
        aclFormat::ACL_FORMAT_ND,
        {DIM_0, DIM_1, 1});
    atb::Tensor gamma = CreateTensorFromVector(
        contextPtr, stream, std::vector<float>(DIM_2, 2.0), ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {DIM_2});
    atb::SVector<atb::Tensor> inTensors = {dy, x, rstd, gamma};
    return inTensors;
}

/**
 * @brief 创建一个rmsnorm backward operation
 * @return atb::Operation * 返回一个Operation指针
 */
atb::Operation *CreateRmsNormBackwardOperation()
{
    atb::train::RmsNormBackwardParam param;
    atb::Operation *rmsNormBackwardOp = nullptr;
    CHECK_STATUS(atb::CreateOperation(param, &rmsNormBackwardOp));
    return rmsNormBackwardOp;
}

int main(int argc, char **argv)
{
    // 设置卡号、创建context、设置stream
    atb::Context *context = nullptr;
    void *stream = nullptr;

    CHECK_STATUS(aclInit(nullptr));
    CHECK_STATUS(aclrtSetDevice(DEVICE_ID));
    CHECK_STATUS(atb::CreateContext(&context));
    CHECK_STATUS(aclrtCreateStream(&stream));
    context->SetExecuteStream(stream);

    // 创建op
```

 版权所有 © 华为技术有限公司

```
    atb::Operation *rmsnormBackwardOp = CreateRmsNormBackwardOperation();
    // 准备输入tensor
    atb::VariantPack variantPack;
    variantPack.inTensors = PrepareInTensor(context, stream);  // 放入输入tensor
    atb::Tensor dx = CreateTensor(ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {DIM_0, DIM_1, DIM_2});
    atb::Tensor dgamma = CreateTensor(ACL_FLOAT, aclFormat::ACL_FORMAT_ND, {DIM_2});
    variantPack.outTensors = {dx, dgamma};  // 放入输出tensor

    uint64_t workspaceSize = 0;
    // 计算workspace大小
    CHECK_STATUS(rmsnormBackwardOp->Setup(variantPack, workspaceSize, context));
    uint8_t *workspacePtr = nullptr;
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtMalloc((void **)(&workspacePtr), workspaceSize,
ACL_MEM_MALLOC_HUGE_FIRST));
    }
    // rmsnorm执行
    rmsnormBackwardOp->Execute(variantPack, workspacePtr, workspaceSize, context);
    CHECK_STATUS(aclrtSynchronizeStream(stream));  // 流同步，等待device侧任务计算完成

    // 释放资源
    for (atb::Tensor &inTensor : variantPack.inTensors) {
        CHECK_STATUS(aclrtFree(inTensor.deviceData));
    }
    for (atb::Tensor &outTensor : variantPack.outTensors) {
        CHECK_STATUS(aclrtFree(outTensor.deviceData));
    }
    if (workspaceSize > 0) {
        CHECK_STATUS(aclrtFree(workspacePtr));
    }
    CHECK_STATUS(atb::DestroyOperation(rmsnormBackwardOp));  // operation，对象概念，先释放
    CHECK_STATUS(aclrtDestroyStream(stream));
    CHECK_STATUS(DestroyContext(context));  // context，全局资源，后释放
    CHECK_STATUS(aclFinalize());
    std::cout << "Rmsnorm backward demo success!" << std::endl;
    return 0;
}
```

11.9 RopeGradOperation

功能

旋转位置编码处理的反向。

算子功能实现描述（Python）

训练算子，实现对于旋转位置编码（RoPE）的梯度反向计算，计算流程如下：

```
def RopeGradCalc(self, in_tensors):
    cos_list = [in_tensors[2][:x, :] for x in OP_PARAM['qSeqLen']]
    sin_list = [in_tensors[3][:x, :] for x in OP_PARAM['qSeqLen']]
    cos = torch.cat(cos_list, dim=0)
    sin = torch.cat(sin_list, dim=0)
    sin1 = sin[:,:64]
    sin2 = sin[:,64:]
    rohqgsin = torch.cat((sin2, -sin1), dim=-1)
    q_grad = torch.zeros_like(in_tensors[0])
    bs = int(in_tensors[0].shape[1] / 128)
    for i in range(bs):
        q_grad[:, i * 128:(i + 1) * 128] = in_tensors[0][:, i * 128:(i + 1) * 128] * (cos + rohqgsin)

    k_grad = torch.zeros_like(in_tensors[1])
    for i in range(bs):
        k_grad[:,i * 128:(i + 1) * 128] = in_tensors[1][:, i * 128:(i + 1) * 128] *(cos + rohqgsin)
        return [q_grad, k_grad]
```

 版权所有 © 华为技术有限公司

定义

```
struct RopeGradParam {
    std::vector<int32_t> qSeqLen;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
qSeqLen	std::vector< int32_t >	-	存储unpad场景下每个batch实际seqlen的值。size不能为0。
rsv[8]	uint8_t	{0}	预留参数。

unpad场景：输入序列长度（ seq length ）是动态的场景下，如果对所有输入序列都按最大长度计算，存在大量冗余计算。Unpad方案是decoder过程序列长度不再按照最大长度计算，而是根据实际的长度进行计算，减少计算量。

输入

参数	维度	数据类型	格式	描述
ropeQ_grad1	[nTokens, hiddenSize]	float16	ND	ropeQ_grad矩阵。
ropeQ_grad2	[nTokens, hiddenSize]	float16	ND	ropeQ_grad矩阵。
cos	[maxSeqLen, head_size]	float16	ND	cos矩阵，maxSeqLen为qSeqLen中的最大元素，head_size为128。
sin	[maxSeqLen, head_size]	float16	ND	sin矩阵。

输出

参数	维度	数据类型	格式	描述
q_grad	[nTokens, hiddenSize]	float16	ND	q_grad矩阵。
k_grad	[nTokens, hiddenSize]	float16	ND	k_grad矩阵。

 版权所有 © 华为技术有限公司

规格约束

- 当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。
- nTokens和qSeqLen需要满足以下关系：

$$nTokens = \sum_{i=0}^{batchsize-1} qSeqLen[i]$$

11.10 StridedBatchMatmulOperation

功能

对矩阵进行分组，指定每组矩阵之间的步长，实现更加灵活的矩阵乘法操作。

定义

```
struct StridedBatchMatmulParam {
    bool transposeA = false;
    bool transposeB = false;
    int32_t batch      = 1;
    int32_t headNum    = 1;
    std::vector<int32_t> m;
    std::vector<int32_t> n;
    std::vector<int32_t> k;
    std::vector<int32_t> lda;
    std::vector<int32_t> ldb;
    std::vector<int32_t> ldc;
    std::vector<int32_t> strideA;
    std::vector<int32_t> strideB;
    std::vector<int32_t> strideC;
    uint8_t rsv[8] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
transposeA	bool	false	是否转置A矩阵。
transposeB	bool	false	是否转置B矩阵。
batch	int32_t	1	batch个数batchSize。
headNum	int32_t	1	多头注意力机制的head数。
m	std::vector< int32_t >	-	A矩阵参与一次矩阵乘指令的shape大小。元素个数为batchSize。
n	std::vector< int32_t >	-	B矩阵参与一次矩阵乘指令的shape大小。元素个数为batchSize。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
k	std::vector< int32_t >	-	C矩阵参与一次矩阵乘指令的shape大小。元素个数为batchSize。
lda	std::vector< int32_t >	-	表示矩阵A的列数。元素个数为batchSize。
ldb	std::vector< int32_t >	-	表示矩阵B的列数。元素个数为batchSize。
ldc	std::vector< int32_t >	-	表示矩阵C的列数。元素个数为batchSize。
strideA	std::vector< int32_t >	-	矩阵A在内存中相邻两次计算之间的跨度。元素个数为batchSize。
strideB	std::vector< int32_t >	-	矩阵B在内存中相邻两次计算之间的跨度。元素个数为batchSize。
strideC	std::vector< int32_t >	-	矩阵C在内存中相邻两次计算之间的跨度。元素个数为batchSize。
rsv[8]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
A	• bmm1、bmm1_grad2、bmm2_grad1：[nTokens, hiddenSize] • bmm1_grad1、bmm2、bmm2_grad2：[nSquareTokens]	float16	ND	输入tensor。
B	• bmm1、bmm1_grad2、bmm2_grad1：[nTokens, hiddenSize] • bmm1_grad1、bmm2、bmm2_grad2：[nSquareTokens]	float16	ND	输入tensor。

输出

参数	维度	数据类型	格式	描述
output	[outdims]	float16	ND	输出tensor。

 版权所有 © 华为技术有限公司

规格约束

当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

11.11 UnpadWithHiddenStateOperation

功能

UnpadWithHiddenState的作用是将填充后的序列恢复成原始的变长序列。

定义

```
struct UnpadWithHiddenStateParam {
    std::vector<int32_t> qSeqLen;
    int32_t maxSeqLen = 4096;
    uint8_t rsv[12] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
qSeqLen	std::vector<int32_t>	-	每个batch实际SeqLen的值，元素个数为batchSize。batchSize的值最大不超过32。
maxSeqLen	int32_t	-	最大SeqLen值。取值不超过4096。
rsv[12]	uint8_t	{0}	预留参数。

输入

参数	维度	数据类型	格式	描述
x	[batchSize, maxSeqLen, hiddenSize]	float16	ND	unpad前，带冗余的tensor，与pad输出shape一致。

输出

参数	维度	数据类型	格式	描述
output	[nTokens, hiddenSize]	float16	ND	unpad后，不带冗余的tensor，与pad输入shape一致。nTokens为qSeqLen数组的元素和。

 版权所有 © 华为技术有限公司

规格约束

当前仅支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

接口调用示例

qSeqlen ={2,3,1} //每个batch的实际输入长度

maxSeqlen=4 //qSeqlen中最大的输入长度

x的shape为（ 3，4，5 ）//batchSize为3，最大输入长度为4，隐藏层大小为5

```
x=[
[1,2,3,4,5
6,7,8,9,10
0,0,0,0,0
0,0,0,0,0],
[11,12,13,14,15
16,17,18,19,20
21,22,23,24,25
0,0,0,0,0],
[26,27,28,29,30
0,0,0,0,0
0,0,0,0,0
0,0,0,0,0]]
```

output的shape为（ 6,5 ） //总tokens数为6 隐藏层大小为5

```
x=[1，2，3，4，5
6，7，8，9，10
11，12，13，14，15
16，17，18，19，20
21，22，23，24，25
26，27，28，29，30]
```

 版权所有 © 华为技术有限公司

12 atb/common_op_params.h

operator==

EventOperation

12.1 operator==

功能

判断参数是否相同。

定义

```
#ifndef ATB_COMMONOPPARAM_H
namespace atb {
namespace common {
inline bool operator==(const EventParam &left, const EventParam &right)
{
    return left.operatorType == right.operatorType && left.event == right.event;
}
} // namespace common
} // namespace atb
#endif
```

12.2 EventOperation

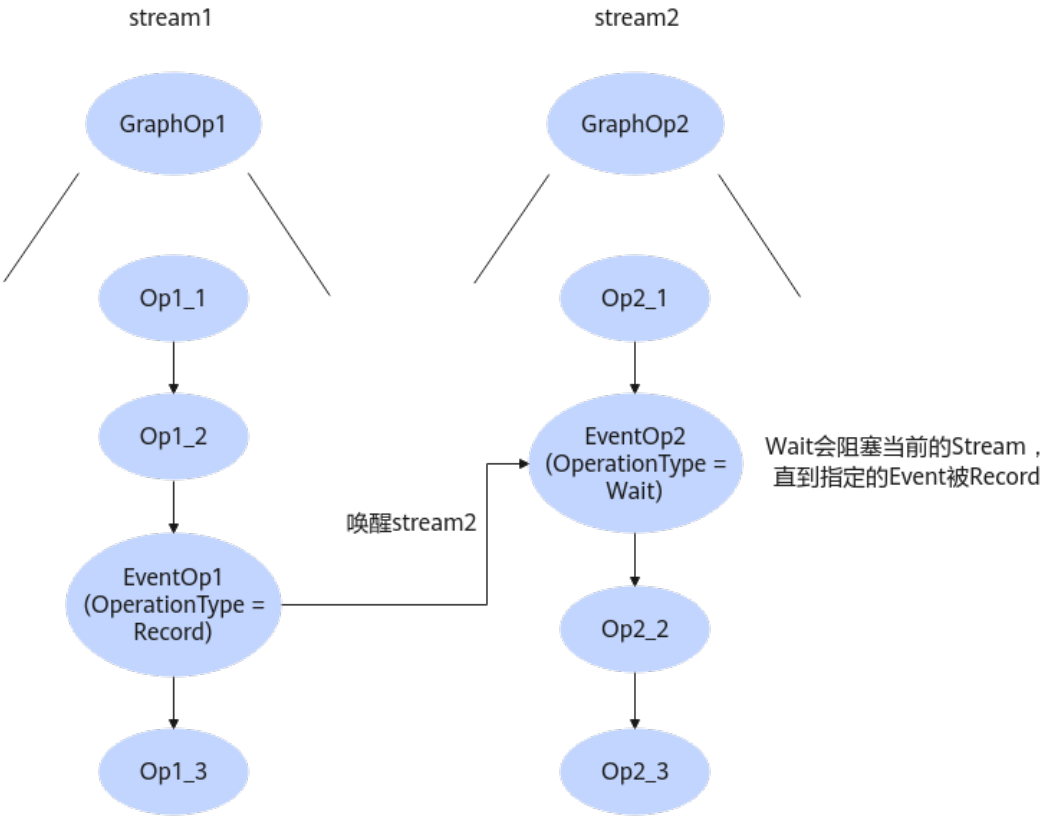
功能

同张卡的**流间同步**功能。Record或者Wait Event。建议使用**aclrtSetOpWaitTimeout**设置等待Event完成的超时时间。

支持先Wait再Record以及先Record再Wait时，建议使用**aclrtCreateEventWithFlag**接口创建Flag为ACL_EVENT_SYNC的Event。

 版权所有 © 华为技术有限公司

算子上下文



定义

```
struct EventParam {
    enum OperatorType : int {
        UNDEFINED = 0,
        RECORD,
        WAIT
    };
    aclrtEvent event;
    OperatorType operatorType = UNDEFINED;
    uint8_t rsv[16] = {0};
};
```

参数列表

成员名称	类型	默认值	描述
OperatorType	int	UNDEFINED	OperatorType支持的值。 • UNDEFINED：默认值，不做任何操作。 • RECORD：在Stream中记录一个Event。 • WAIT：阻塞指定Stream的运行，直至指定的Event完成。
event	aclrtEvent	-	需要RECORD或者WAIT的Event。
operatorType	OperatorType	UNDEFINED	OperatorType，支持UNDEFINED、RECORD和WAIT。

 版权所有 © 华为技术有限公司

成员名称	类型	默认值	描述
rsv[16]	uint8_t	{0}	预留参数。

功能列表

- **跳过功能**

  当operatorType设置为UNDEFINED时，调用Execute接口时，将不会进行任何操作。

- **Record功能**

  调用Execute接口时，会调用aclrtRecordEvent接口在stream中记录EventOperation的Param中传入的aclrtEvent。

  ```
  atb::common::EventParam param;
  param.operatorType = atb::common::EventParam::OperatorType::RECORD;
  aclrtEvent event = nullptr;
  param.event = event;
  atb::Operation *operation = nullptr;
  Status st = atb::CreateOperation(param, &operation);
  ```

- **Wait功能**

  调用Execute接口时，会调用aclrtStreamWaitEvent接口和aclrtResetEvent接口。会先阻塞该Operation所使用的Stream的运行，直到使用Record功能记录该event，然后再复位该event，使得该event可以反复被Record或者Wait（反复的被EventOperation所使用）。

  ```
  atb::common::EventParam param;
  param.operatorType = atb::common::EventParam::OperatorType::WAIT;
  aclrtEvent event;
  aclrtCreateEventWithFlag(&event, ACL_EVENT_SYNC);
  param.event = event;
  atb::Operation *operation = nullptr;
  atb::CreateOperation(param, &operation);
  ```

 版权所有 © 华为技术有限公司

13 atb/graph_op_builder.h

GraphOpBuilder
CreateGraphOpBuilder
DestroyGraphOpBuilder

13.1 GraphOpBuilder

13.1.1 GraphOpBuilder

功能

构造函数。

定义

```
GraphOpBuilder()
```

返回值

无。

13.1.2 ~GraphOpBuilder

功能

析构函数。

定义

```
virtual ~GraphOpBuilder()
```

返回值

无。

 版权所有 © 华为技术有限公司

13.1.3 Init

功能

初始化图算子。

定义图算子的名称，输入输出及shape传导规则。

定义

```
virtual Status Init (const std::string &opName, const InferShapeFunc &inferShapeFunc, const SVector<std::string> &inTensorNames, const SVector<std::string> &outTensorNames)=0
```

成员

成员名称	描述
opName	图算子的名称。
inferShapeFunc	图算子的shape传导规则。
inTensorNames	输入tensor名称。
outTensorNames	输出tensor名称。

返回值

状态值。如果设置成功，返回NO_ERROR。

13.1.4 Reshape

功能

改变输入tensor的shape。

定义

```
virtual Status Reshape (const std::string &srcTensorName, const ReshapeFunc &reshapeFunc, const std::string &viewTensorName)=0
```

成员

成员名称	描述
srcTensorName	输入tensor的名称。
reshapeFunc	shape修改规则。
viewTensorName	shape修改后的tensor名称。

 版权所有 © 华为技术有限公司

返回值

状态值。如果设置成功，返回NO_ERROR。

13.1.5 AddOperation

功能

- 向图中添加算子。
- 创建并向图中添加算子。

定义

```
virtual Status AddOperation (Operation *operation, const SVector< std::string > &inTensorNames, const
SVector< std::string > &outTensorNames)=0;

template <class OpParam>
Status AddOperation(const OpParam &opParam, const SVector<std::string> &inTensorNames, const
SVector< std::string > &outTensorNames)
{
    Operation *operation = nullptr;
    Status st = CreateOperation(opParam, &operation);
    if (st != NO_ERROR) {
        return st;
    }
    st =  AddOperation(operation, inTensorNames, outTensorNames);
    if (st != NO_ERROR) {
        if (operation != nullptr) {
            DestroyOperation(operation);
        }
    }
    return st;
}
```

成员

成员名称	描述
opParam	要添加的算子参数。
inTensorNames	被添加算子的输入tensor名称。
outTensorNames	被添加算子的输出tensor名称。

返回值

状态值。如果设置成功，返回NO_ERROR。

13.1.6 Build

功能

创建图算子。

定义

```
virtual Operation * Build ()=0
```

 版权所有 © 华为技术有限公司

返回值

返回被创建的图算子，失败返回空指针。

13.2 CreateGraphOpBuilder

功能

创建图算子构建器。

定义

```
Status CreateGraphOpBuilder (GraphOpBuilder **builder)
```

成员

成员名称	描述
builder	返回被创建的构建器。

返回值

状态值。如果设置成功，返回NO_ERROR。

13.3 DestroyGraphOpBuilder

功能

销毁图算子构建器。

定义

```
Status atb::DestroyGraphOpBuilder (GraphOpBuilder *builder)
```

成员

成员名称	描述
builder	待销毁的构建器。

返回值

状态值。如果设置成功，返回NO_ERROR。

 版权所有 © 华为技术有限公司

14 atb/utils.h

Utils

14.1 Utils

功能

加速库公共工具接口类。

定义

```
class Utils {
public:
    static std::string GetAtbVersion();
    static uint64_t GetTensorSize(const Tensor &tensor);
    static uint64_t GetTensorSize(const TensorDesc &tensorDesc);
    static uint64_t GetTensorNumel(const Tensor &tensor);
    static uint64_t GetTensorNumel(const TensorDesc &tensorDesc);
    static void QuantParamConvert(const float *src, uint64_t *dest, uint64_t itemCount);
};
```

成员

成员名称	描述
GetAtbVersion	返回加速库版本信息。
GetTensorSize	返回Tensor对象的数据存储大小。
GetTensorNumel	返回Tensor对象的数据个数。
QuantParamConvert	量化场景使用。float数组转成uint64数组，实现逻辑是复制float到uint64的后32位，uint64的前32位置0。 • src：输入float数组。 • dest：转化得到的uint64数组。 • itemCount：数组元素个数。

 版权所有 © 华为技术有限公司

15 atb/comm.h

CreateHcclComm

CreateHcclCommByRankTableFile

DestoryHcclComm

CreateHcclCrossMulitComm

15.1 CreateHcclComm

功能

backend为HCCL时，rootInfo方式创建单机多卡通信域功能。

定义

```
namespace atb {
using void *HcclComm;
namespace Comm {
HcclComm CreateHcclComm(int32_t rank, int32_t rankRoot, int32_t rankSize, char *commName);
}; // namespace Comm
} // namespace atb
```

参数列表

成员名称	输入/输出	类型	默认值	描述
rank	输入	int32_t	-	当前卡所属通信编号。
rankRoot	输入	int32_t	-	主通信编号。
rankSize	输入	int32_t	-	通信的卡的数量。
commName	输出	char *	-	通信域名称。

 版权所有 © 华为技术有限公司

规格约束

rank、rankSize、rankRoot需满足以下条件。

- 0 ≤ rank < rankSize
- 0 ≤ rankRoot < rankSize

只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

15.2 CreateHcclCommByRankTableFile

功能

backend为HCCL时，使用ranktable的方式创建单机多卡通信域功能。

定义

```
namespace atb {
using void *HcclComm;
namespace Comm {
HcclComm CreateHcclCommByRankTableFile(int32_t rank, int32_t rankSize, const char *rankTableFile, char
*commName);
}; // namespace Comm
} // namespace atb
```

参数列表

成员名称	输入/输出	类型	默认值	描述
rank	输入	int32_t	-	当前卡所属通信编号。
rankSize	输入	int32_t	-	通信的卡的数量。
rankTableFile	输入	char *	-	集群信息的配置文件路径。
commName	输出	char *	-	通信域名称。

规格约束

rank、rankSize、rankRoot需满足0 ≤ rank < rankSize。

只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

15.3 DestoryHcclComm

功能

backend为HCCL时，销毁通信域功能。

 版权所有 © 华为技术有限公司

定义

```
namespace atb {
using void *HcclComm;
namespace Comm {
Status DestoryHcclComm(HcclComm comm);
}; // namespace Comm
} // namespace atb
```

参数列表

成员名称	输入/输出	类型	默认值	描述
hcclComm	输入	HcclComm	nullptr	HCCL通信域指针。

规格约束

只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

15.4 CreateHcclCrossMulitComm

功能

backend为HCCL时，使用ranktable的方式创建多机多卡子通信域功能。

定义

```
HcclComm CreateHcclCrossMulitComm(const char *rankTableFile, uint32_t subCommRankId,
std::vector<uint32_t> &rankIds, uint64_t subCommId, uint32_t hcclBufferSize, char *commName);
```

参数列表

成员名称	输入/输出	类型	默认值	描述
rankTableFile	输入	char *	-	ranktable的文件路径。作为字符串最大长度为4096字节，含结束符。 ranktable的配置请参见《TensorFlow 1.15模型迁移指南》的“模型训练>执行分布式训练>**准备ranktable资源配置文件**”章节。
subCommRankId	输入	uint32_t	-	本rank在子通信域中的rank id。
rankIds	输入	vector<uint32_t>	-	子通信域中rank在全局通信域中的rank id组成的数组。 注意：该数组应当是有序的，数组中每个rank的下标将映射为其在子通信域的rank id。

 版权所有 © 华为技术有限公司

成员名称	输入/输出	类型	默认值	描述
subCommId	输入	uint64_t	-	当前子通信域标识。
hcclBufferSize	输入	uint32_t	-	子通信域共享数据的缓存区大小。单位M
commName	输出	char *	-	子通信域创建成功后获取到的通信域名称。最大返回长度支持128。

规格约束

- 当前rank不支持在多个子通信域中并行创建，如：当前rank是1，两个子通信域分别是[0,1]和[1,2]，此时不支持并行创建，可以串行创建；如果当前rank是0或2，两个子通信域分别是[0,1]和[1,2]，此时支持并行创建。
- subCommRankId < rankIds.size()。
- 不支持跨芯片类型多机通信。
- 对于同一个全局通信域，不可使用重复的subCommId创建子通信域。
- 只支持Atlas 800I A2 推理产品/Atlas A2 训练系列产品。

 版权所有 © 华为技术有限公司

16 ATB Python API 使用说明

OpParam

Operation

16.1 OpParam

16.1.1 说明

目前支持的Param有LayerNormParam、ElewiseParam、LinearParam、SoftmaxParam、RopeParam、SelfAttentionParam、SplitParam、GatherParam、ActivationParam、RmsNormParam。

枚举项、param的属性值、结构体等定义和C++保持一致。

16.1.2 AclDataType

枚举项

- ACL_DT_UNDEFINED
- ACL_FLOAT
- ACL_FLOAT16
- ACL_INT8
- ACL_INT32
- ACL_UINT8
- ACL_INT16
- ACL_UINT16
- ACL_UINT32
- ACL_INT64
- ACL_UINT64
- ACL_DOUBLE
- ACL_BOOL

 版权所有 © 华为技术有限公司

- ACL_STRING
- ACL_COMPLEX64
- ACL_COMPLEX128
- ACL_BF16
- ACL_INT4
- ACL_UINT1
- ACL_COMPLEX32

使用示例：

```
torch_atb.AclDataType.ACL_DT_UNDEFINED
```

16.1.3 AclFormat

枚举项

- ACL_FORMAT_UNDEFINED
- ACL_FORMAT_NCHW
- ACL_FORMAT_NHWC
- ACL_FORMAT_ND
- ACL_FORMAT_NC1HWC0
- ACL_FORMAT_FRACTAL_Z
- ACL_FORMAT_NC1HWC0_C04
- ACL_FORMAT_HWCN
- ACL_FORMAT_NDHWC
- ACL_FORMAT_FRACTAL_NZ
- ACL_FORMAT_NCDHW
- ACL_FORMAT_NDC1HWC0
- ACL_FRACTAL_Z_3D

使用示例：

```
torch_atb.AclFormat.ACL_FORMAT_UNDEFINED
```

16.1.4 QuantType

枚举项

QUANT_UNQUANT

QUANT_INT4

QUANT_INT8

QUANT_INT16

QUANT_FLOAT8

QUANT_FLOAT16

 版权所有 © 华为技术有限公司

使用示例：

```
torch_atb.QuantType.QUANT_UNQUANT
```

16.1.5 DynamicQuantType

枚举项

- DYNAMIC_QUANT_UNDEFINED
- DYNAMIC_QUANT_SYMMETRIC
- DYNAMIC_QUANT_ASYMMETRIC

使用示例：

```
torch_atb.DynamicQuantType.DYNAMIC_QUANT_UNDEFINED
```

16.1.6 ActivationType

枚举项

- ACTIVATION_UNDEFINED
- ACTIVATION_RELU
- ACTIVATION_GELU
- ACTIVATION_FAST_GELU
- ACTIVATION_SWISH
- ACTIVATION_LOG
- ACTIVATION_SWIGLU_FORWARD
- ACTIVATION_SWIGLU_BACKWARD
- ACTIVATION_SIGMOID
- ACTIVATION_FASTER_GELU_FORWARD
- ACTIVATION_MAX

使用示例：

```
torch_atb.ActivationType.ACTIVATION_UNDEFINED
```

16.1.7 InputLayout

枚举项

- TYPE_BSND
- TYPE_BNSD

 版权所有 © 华为技术有限公司

16.1.8 ActivationParam

属性	类型	默认值	描述
activation_type	torch_atb.ActivationType	torch_atb.ActivationType.ACTIVATION_UNDEFINED	此默认类型不可用，用户需配置此项参数。
scale	float	1.0	-
dim	int	-1	-
gelu_mode	torch_atb.ActivationParam.GeLUMode	torch_atb.ActivationParam.GeLUMode.TANH_MODE	-

ActivationParam.GeLUMode

枚举项：

- TANH_MODE
- NONE_MODE

调用示例

```
import torch
import torch_atb

def activation():
    activation_param = torch_atb.ActivationParam(
        activation_type = torch_atb.ActivationType.ACTIVATION_SWISH, scale = 1.0)
    activation = torch_atb.Operation(activation_param)

    intensor = torch.rand(2, 3, 5).bfloat16()
    intensor_npu = intensor.npu()
    print("intensor: ", intensor_npu)

    def activation_run():
        activation_outputs = activation.forward([intensor_npu])
        return activation_outputs

    outputs = activation_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    activation()
```

16.1.9 ElewiseParam

属性	类型	默认值	描述
elewise_type	torch_atb.ElewiseParam.ElewiseType	torch_atb.ElewiseParam.ElewiseType.ELEWISE_UNDEFINED	此默认类型不可用，用户需配置此项参数。
quant_param	torch_atb.ElewiseParam.QuantParam	-	-

 版权所有 © 华为技术有限公司

属性	类型	默认值	描述
muls_param	torch_atb.ElewiseParam. MulsParam	-	-
out_tensor_type	torch_atb.AclDataType	torch_atb.AclDataType.ACL_DT_UNDEFINED	根据输入tensors自动推导输出tensors数据类型。

ElewiseParam.ElewiseType

枚举项：

- ELEWISE_UNDEFINED
- ELEWISE_CAST
- ELEWISE_MULS
- ELEWISE_COS
- ELEWISE_SIN
- ELEWISE_SUB
- ELEWISE_EQUAL
- ELEWISE_NEG
- ELEWISE_QUANT
- ELEWISE_LOGICAL_NOT
- ELEWISE_ADD
- ELEWISE_MUL
- ELEWISE_QUANT_PER_CHANNEL
- ELEWISE_DEQUANT_PER_CHANNEL
- ELEWISE_REALDIV
- ELEWISE_LOGICAL_AND
- ELEWISE_LOGICAL_OR
- ELEWISE_LESS
- ELEWISE_GREATER
- ELEWISE_DYNAMIC_QUANT
- ELEWISE_TANH

ElewiseParam.QuantParam

属性	类型	默认值	描述
input_scale	float	1.0	-
asymmetric	bool	False	-

 版权所有 © 华为技术有限公司

属性	类型	默认值	描述
input_offset	int	0	-

ElewiseParam.MulsParam

属性	类型	默认值	描述
var_attr	float	0.0	-

调用示例

```
import torch
import torch_atb

def elewise_add():
    input1_npu = torch.randn(2, 3, dtype=torch.float16).npu()
    input2_npu = torch.randn(2, 3, dtype=torch.float16).npu()
    print("input1: ", input1_npu)
    print("input2: ", input2_npu)
    elewise_param = torch_atb.ElewiseParam(elewise_type =
torch_atb.ElewiseParam.ElewiseType.ELEWISE_ADD)
    elewise = torch_atb.Operation(elewise_param)

    def elewise_run():
        elewise_outputs = elewise.forward([input1_npu, input2_npu])
        return elewise_outputs

    outputs = elewise_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    elewise_add()
```

16.1.10 GatherParam

属性	类型	默认值	描述
axis	int	0	-
batch_dims	int	0	-

调用示例

```
import torch
import torch_atb

def gather():
    gather_param = torch_atb.GatherParam(axis = 1)
    gather = torch_atb.Operation(gather_param)
    intensor0 = torch.randn([3,5],dtype=torch.float16)
    intensor0_npu = intensor0.npu()
    print("intensor0: ", intensor0)
    intensor1 = torch.randint(0, 5, [3,4],dtype=torch.int64)
    intensor1_npu = intensor1.npu()
    print("intensor1: ", intensor1)
```

 版权所有 © 华为技术有限公司

```
    def gather_run():
        gather_outputs = gather.forward([intensor0_npu, intensor1_npu])
        return gather_outputs

    outputs = gather_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    gather()
```

16.1.11 LayerNormParam

属性	类型	默认值	描述
layer_type	torch_atb.LayerNormParam.LayerNormType	torch_atb.LayerNormParam.LayerNormType.LAYER_NORM_UNDEFINED	此默认类型不可用，用户需配置此项参数。
norm_param	torch_atb.LayerNormParam.NormParam	-	-
pre_norm_param	torch_atb.LayerNormParam.PreNormParam	-	-
post_norm_param	torch_atb.LayerNormParam.PostNormParam	-	-

LayerNormParam.LayerNormType

枚举项：

- LAYER_NORM_UNDEFINED
- LAYER_NORM_NORM
- LAYER_NORM_PRENORM
- LAYER_NORM_POSTNORM
- LAYER_NORM_MAX

LayerNormParam.NormParam

属性	类型	默认值	描述
quant_type	torch_atb.QuantType	torch_atb.QuantType.QUANT_UNQUANT	表示不进行量化操作。
epsilon	float	1e-5	-

 版权所有 © 华为技术有限公司

属性	类型	默认值	描述
begin_norm_axis	int	0	-
begin_params_axis	int	0	-
dynamic_quant_type	torch_atb.DynamicQuantType	torch_atb.DynamicQuantType.DYNAMIC_QUANT_UNDEFINED	-

LayerNormParam.PreNormParam

属性	类型	默认值	描述
quant_type	torch_atb.QuantType	torch_atb.QuantType.QUANT_UNQUANT	表示不进行量化操作。
epsilon	float	1e-5	-
op_mode	int	0	-
zoom_scale_value	float	1.0	-

LayerNormParam.PostNormParam

属性	类型	默认值	描述
quant_type	torch_atb.QuantType	torch_atb.QuantType.QUANT_UNQUANT	表示不进行量化操作。
epsilon	float	1e-5	-
op_mode	int	0	-
zoom_scale_value	float	1.0	-

调用示例

```
import torch
import torch_atb
import numbers

def layernrom():
    eps=1e-05
    batch, sentence_length, embedding_dim = 20, 5, 10
```

 版权所有 © 华为技术有限公司

```
    embedding = torch.randn(batch, sentence_length, embedding_dim)
    embedding_npu = embedding.npu()
    normalized_shape = (embedding_dim,) if isinstance(embedding_dim, numbers.Integral) else
tuple(embedding_dim)
    weight = torch.ones(normalized_shape, dtype=torch.float32).npu()
    bias = torch.zeros(normalized_shape, dtype=torch.float32).npu()
    print("embedding: ", embedding_npu)
    print("weight: ", weight)
    print("bias: ", bias)
    layer_norm_param = torch_atb.LayerNormParam(layer_type =
torch_atb.LayerNormParam.LayerNormType.LAYER_NORM_NORM)
    layer_norm_param.norm_param.epsilon = eps
    layer_norm_param.norm_param.begin_norm_axis = len(normalized_shape) * -1
    layernorm = torch_atb.Operation(layer_norm_param)

    def layernorm_run():
        layernorm_outputs = layernorm.forward([embedding_npu, weight, bias])
        return layernorm_outputs

    outputs = layernorm_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    layernrom()
```

16.1.12 LinearParam

属性	类型	默认值	描述
transpose_a	bool	False	-
transpose_b	bool	True	-
has_bias	bool	True	-
out_data_type	torch_atb.AclDataType	torch_atb.AclDataType.ACL_DT_UNDEFINED	根据输入tensors自动推导输出tensors数据类型。
en_accum	bool	False	-
matmul_type	torch_atb.LinearParam.MatmulType	torch_atb.LinearParam.MatmulType.MATMUL_UNDEFINED	-

LinearParam.MatmulType

枚举项：

- MATMUL_UNDEFINED
- MATMUL_EIN_SUM

调用示例

```
import torch
import torch_atb

def linear():
    m, k, n = 3, 4, 5
```

 版权所有 © 华为技术有限公司

```
    input_tensor = torch.randn(m, k, dtype=torch.float16).npu()
    weight_tensor = torch.randn(k, n, dtype=torch.float16).npu()
    print(input_tensor)
    print(weight_tensor)

    linear_param = torch_atb.LinearParam(has_bias = False, transpose_b = False)
    linear = torch_atb.Operation(linear_param)

    def linear_run():
        linear_outputs = linear.forward([input_tensor, weight_tensor])
        return [linear_outputs[0].to(torch.float32)]


    outputs = linear_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    linear()
```

16.1.13 RopeParam

属性	类型	默认值	描述
rotary_coeff	int	4	-
cos_format	int	0	-

调用示例

```
import torch
import torch_atb

def rope():
    rope_param = torch_atb.RopeParam(rotary_coeff = 4)
    rope = torch_atb.Operation(rope_param)

    def gen_inputs():
        ntoken = 4
        seqlen = 4
        hidden_size = 4096
        head_size = 128
        intensor0 = torch.rand(ntoken, hidden_size).half()
        intensor1 = torch.rand(ntoken, hidden_size).half()
        intensor2 = torch.rand(ntoken, head_size).half()
        intensor3 = torch.rand(ntoken, head_size).half()
        intensor4 = torch.tensor([seqlen], dtype=torch.int32)
        return [intensor0, intensor1, intensor2, intensor3, intensor4]

    in_tensors = gen_inputs()
    in_tensors_npu = [tensor.npu() for tensor in in_tensors]
    print("in_tensors: ", in_tensors_npu)

    def rope_run():
        rope_outputs = rope.forward(in_tensors_npu)
        return rope_outputs

    outputs = rope_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    rope()
```

 版权所有 © 华为技术有限公司

16.1.14 SelfAttentionParam

属性	类型	默认值	描述
quant_type	torch_atb.SelfAttentionParam.QuantType	torch_atb.QuantType.TYPE_QUANT_UNQUANT	表示不进行量化操作。
out_data_type	torch_atb.AclDataType	torch_atb.AclDataType.ACL_DT_UNDEFINED	根据输入tensors自动推导输出tensors数据类型。
head_num	int	0	此默认值不可用，用户需配置此项参数。
kv_head_num	int	0	-
q_scale	float	1.0	-
qk_scale	float	1.0	-
batch_run_status_enable	bool	False	-
is_triu_mask	int	0	-
calc_type	torch_atb.SelfAttentionParam.CalcType	torch_atb.SelfAttentionParam.CalcType.UNDEFINED	decoder&encoder for flashAttention。
kernel_type	torch_atb.SelfAttentionParam.KernelType	torch_atb.SelfAttentionParam.KernelType.KERNELTYPE_DEFAULT	-
clamp_type	torch_atb.SelfAttentionParam.ClampType	torch_atb.SelfAttentionParam.ClampType.CLAMP_TYPE_UNDEFINED	不做clamp。
clamp_min	float	0.0	-
clamp_max	float	0.0	-
mask_type	torch_atb.SelfAttentionParam.MaskType	torch_atb.SelfAttentionParam.MaskType.MASK_TYPE_UNDEFINED	全0mask。
kvcache_cfg	torch_atb.SelfAttentionParam.KvCacheCfg	torch_atb.SelfAttentionParam.KvCacheCfg.K_CACHE_V_CACHE	-

 版权所有 © 华为技术有限公司

属性	类型	默认值	描述
scale_type	torch_atb.SelfAttentionParam.ScaleType	torch_atb.SelfAttentionParam.ScaleType.SCALE_TYPE_TOR	-
input_layout	torch_atb.InputLayout	torch_atb.InputLayout.TYPE_BSND	-
mla_v_head_size	int	0	-
cache_type	torch_atb.SelfAttentionParam.CacheType	torch_atb.SelfAttentionParam.CacheType.CACHE_TYPE_NORM	-
window_size	int	0	-

SelfAttentionParam.QuantType

枚举项：

- TYPE_QUANT_UNQUANT
- TYPE_DEQUANT_FUSION
- TYPE_QUANT_QKV_OFFLINE
- TYPE_QUANT_QKV_ONLINE

SelfAttentionParam.CalcType

枚举项：

- UNDEFINED
- ENCODER
- DECODER
- PA_ENCODER
- PREFIX_ENCODER

SelfAttentionParam.KernelType

枚举项：

- KERNELTYPE_DEFAULT
- KERNELTYPE_HIGH_PRECISION

SelfAttentionParam.ClampType

枚举项：

 版权所有 © 华为技术有限公司

- CLAMP_TYPE_UNDEFINED
- CLAMP_TYPE_MIN_MAX

SelfAttentionParam.MaskType

枚举项：

- MASK_TYPE_UNDEFINED
- MASK_TYPE_NORM
- MASK_TYPE_ALIBI
- MASK_TYPE_NORM_COMPRESS
- MASK_TYPE_ALIBI_COMPRESS
- MASK_TYPE_ALIBI_COMPRESS_SQRT
- MASK_TYPE_ALIBI_COMPRESS_LEFT_ALIGN
- MASK_TYPE_SLIDING_WINDOW_NORM
- MASK_TYPE_SLIDING_WINDOW_COMPRESS

SelfAttentionParam.KvCacheCfg

枚举项：

- K_CACHE_V_CACHE
- K_BYPASS_V_BYPASS

SelfAttentionParam.ScaleType

枚举项：

- SCALE_TYPE_TOR
- SCALE_TYPE_LOGN
- SCALE_TYPE_MAX

SelfAttentionParam.CacheType

枚举项：

- CACHE_TYPE_NORM
- CACHE_TYPE_SWA

调用示例

```
import torch
import torch_atb

def self_attention():
    self_attention_param = torch_atb.SelfAttentionParam(head_num = 24, kv_head_num = 24)
    self_attention_param.calc_type = torch_atb.SelfAttentionParam.CalcType.PA_ENCODER
    self_attention = torch_atb.Operation(self_attention_param)
    q = torch.ones(4096, 24, 64, dtype=torch.float16).npu()
    k = torch.ones(4096, 24, 64, dtype=torch.float16).npu()
    v = torch.ones(4096, 24, 64, dtype=torch.float16).npu()
    seqlen = torch.tensor([4096], dtype=torch.int32)
```

 版权所有 © 华为技术有限公司

```
    intensors = [q,k,v,seqlen]
    print("intensors: ", intensors)

    def self_attention_run():
        outputs = self_attention.forward([q,k,v,seqlen])
        return [outputs]

    outputs = self_attention_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    self_attention()
```

16.1.15 RmsNormParam

属性	类型	默认值	描述
layer_type	torch_atb.RmsNormParam.RmsNormType	torch_atb.RmsNormParam.RmsNormType.RMS_NORM_UNDEFINED	此默认类型不可用，用户需配置此项参数。
norm_param	torch_atb.RmsNormParam.NormParam	-	-
pre_norm_param	torch_atb.RmsNormParam.PreNormParam	-	-
post_norm_param	torch_atb.RmsNormParam.PostNormParam	-	-

RmsNormParam.RmsNormType

枚举项：

- RMS_NORM_UNDEFINED
- RMS_NORM_PRENORM
- RMS_NORM_NORM
- RMS_NORM_POSTNORM

RmsNormParam.NormParam

属性	类型	默认值	描述
quant_type	torch_atb.QuantType	torch_atb.QuantType.QUANT_UNQUANT	表示不进行量化操作。
epsilon	float	1e-5	-
layer_norm_eps	double	1e-5	-
rstd	bool	False	-

 版权所有 © 华为技术有限公司

属性	类型	默认值	描述
precision_mode	torch_atb.RmsNormParam.PrecisionMode	torch_atb.RmsNormParam.PrecisionMode.HIGH_PRECISION_MODE	-
model_type	torch_atb.RmsNormParam.ModelType	torch_atb.RmsNormParam.ModelType.LLAMA_MODEL	-
dynamic_quant_type	torch_atb.DynamicQuantType	torch_atb.DynamicQuantType.DYNAMIC_QUANT_UNDEFINED	-

RmsNormParam.PrecisionMode

枚举项：

- HIGH_PRECISION_MODE
- HIGH_PERFORMANCE_MODE

RmsNormParam.ModelType

枚举项：

- LLAMA_MODEL
- GEMMA_MODEL

RmsNormParam.PreNormParam

属性	类型	默认值	描述
quant_type	torch_atb.QuantType	torch_atb.QuantType.QUANT_UNQUANT	表示不进行量化操作。
epsilon	float	1e-5	-
has_bias	bool	False	-

RmsNormParam.PostNormParam

属性	类型	默认值	描述
quant_type	torch_atb.QuantType	torch_atb.QuantType.QUANT_UNQUANT	表示不进行量化操作。
epsilon	float	1e-5	-

 版权所有 © 华为技术有限公司

属性	类型	默认值	描述
has_bias	bool	False	-

调用示例

```
import torch
import torch_atb
import numpy as np

def rms_norm():
    rms_norm_param = torch_atb.RmsNormParam(layer_type =
torch_atb.RmsNormParam.RmsNormType.RMS_NORM_NORM)
    rms_norm_param.norm_param.rstd = True
    rms_norm = torch_atb.Operation(rms_norm_param)
    shape=[8, 8, 8]
    shape_gamma=[8]
    x = torch.from_numpy(np.random.uniform(low=0, high=100, size=shape).astype(np.float32))
    gamma = torch.from_numpy(np.random.uniform(low=0, high=100,
size=shape_gamma).astype(np.float32))
    in_tensors = [x.npu(), gamma.npu()]
    print("in_tensors: ", in_tensors)

    def rms_norm_run():
        rms_norm_outputs = rms_norm.forward(in_tensors)
        return rms_norm_outputs

    outputs = rms_norm_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    rms_norm()
```

16.1.16 SoftmaxParam

属性	类型	默认值	描述
axes	List[int]	list()	-

约束说明

axes的长度大小需小于**DEFAULT_SVECTOR_SIZE**，否则会抛出长度溢出异常。

调用示例

```
import torch
import torch_atb

def softmax():
    input_npu = torch.randn(2, 16, 256, dtype=torch.float32).npu()
    print("input: ", input_npu)
    softmax_param = torch_atb.SoftmaxParam(axes = [0])
    softmax = torch_atb.Operation(softmax_param)

    def softmax_run():
        softmax_outputs = softmax.forward([input_npu])
        return softmax_outputs
```

 版权所有 © 华为技术有限公司

```
    outputs = softmax_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    softmax()
```

16.1.17 SplitParam

属性	类型	默认值	描述
split_dim	int	0	-
split_num	int	2	-
split_sizes	List[int]	list()	-

约束说明

split_sizes的长度大小需小于**DEFAULT_SVECTOR_SIZE**，否则会抛出长度溢出异常。

调用示例

```
import torch
import torch_atb

def split():
    input_npu = torch.randn(6, 6, dtype=torch.float16).npu()
    print("input: ", input_npu)
    split_param = torch_atb.SplitParam(split_dim = 0, split_num = 2)
    split = torch_atb.Operation(split_param)

    def split_run():
        split_outputs = split.forward([input_npu])
        return split_outputs

    outputs = split_run()
    print("outputs: ", outputs)

if __name__ == "__main__":
    split()
```

16.2 Operation

针对用户调用单OP的场景，Operation提供创建OP、执行OP、OP信息调试等接口。可以直接通过该对象完成OP相关的操作。

创建 Operation

接口名称	CLASS torch_atb.Operation(OpParam)
接口描述	创建单OP
Input要求/参数（可选）	OpParam
Output要求/参数（可选）	torch_atb.Operation

 版权所有 © 华为技术有限公司

目前支持的入参类型有：LayerNormParam、ElewiseParam、LinearParam、SoftmaxParam、RopeParam、SelfAttentionParam、PagedAttentionParam、SplitParam、ReshapeAndCacheParam、GatherParam、ActivationParam、RmsNormParam。

name

功能	torch_atb.Operation的属性，用于获取torch_atb.Operation的名称。
原型	torch_atb.Operation.name -> str
参数	NA
返回值	Operation的名称。
注意	调用接口失败，会抛出异常终止运行。

使用示例：

```
linear = torch_atb.Operation(linear_param)
print(linear.name)
```

input_num

功能	torch_atb.Operation的属性，用于获取输入tensor的个数。
原型	torch_atb.Operation.input_num ->int
参数	NA
返回值	输入tensor的个数。
注意	调用接口失败，会抛出异常终止运行。

使用示例：

```
linear = torch_atb.Operation(linear_param)
print(linear.input_num)
```

output_num

功能	torch_atb.Operation的属性，用于获取输入tensor的个数。
原型	torch_atb.Operation.output_num ->int
参数	NA
返回值	输出tensor的个数。

 版权所有 © 华为技术有限公司

注意	调用接口失败，会抛出异常终止运行。

使用示例：

```
linear = torch_atb.Operation(linear_param)
print(linear.output_num)
```

forward

功能	torch_atb.Operation的方法，用于执行Operation。
原型	torch_atb.Operation.forward(List:torch::Tensor) -> List:torch::Tensor
参数	输入tensor，类型为List[torch.Tensor]。
返回值	输出tensor，类型为List[torch.Tensor]。
注意	调用接口失败，会抛出异常终止运行。

使用示例：

```
input = torch.randn(m, k, dtype=torch.float16).npu()
weight = torch.randn(k, n, dtype=torch.float16).npu()
linear_param = torch_atb.LinearParam()
linear_param.has_bias = False
linear_param.transpose_b = False
linear = torch_atb.Operation(linear_param)
linear_outputs = linear.forward([input, weight])
print("linear_outputs : ", linear_outputs )
```

 版权所有 © 华为技术有限公司

17 术语&缩略语

术语/缩略语	含义
batch	• 单样本推理： 如果每次推理只传入一个样本，这种方式称为单样本推理（单独处理输入）。 示例：输入一张图片或一个句子，让模型预测结果。 • 批量推理： 如果一次传入多个样本组成的批量（batch），模型会同时对多个样本进行推理。 示例：输入 32 张图片或 32 个句子，让模型并行预测所有结果。
prof/ profiling	测量算子的执行细节，包括调用栈，调用时间等，使用**msprof**采集，用于性能分析和调优。
tiling	大多数情况下，Local Memory的存储，无法完整的容纳算子的输入与输出，需要每次搬运一部分输入进行计算然后搬出，再搬运下一部分输入进行计算，直到得到完整的最终结果，这个数据切分、分块计算的过程称之为Tiling。
算子	算子是一种逻辑运行单元。机器学习服务将各种数据加载、数据预处理、机器学习算法等处理逻辑封装成不同的逻辑运行单元，以便灵活调度。 算子也可称为执行算子，是流计算系统中可被调度执行计算任务的最小单元。

 版权所有 © 华为技术有限公司