PyAsc Is All You Need
——高性能算子开发新范式

作者：苏统华（微信：su-tonghua）
单位：哈尔滨工业大学
时间：2025年12月2日

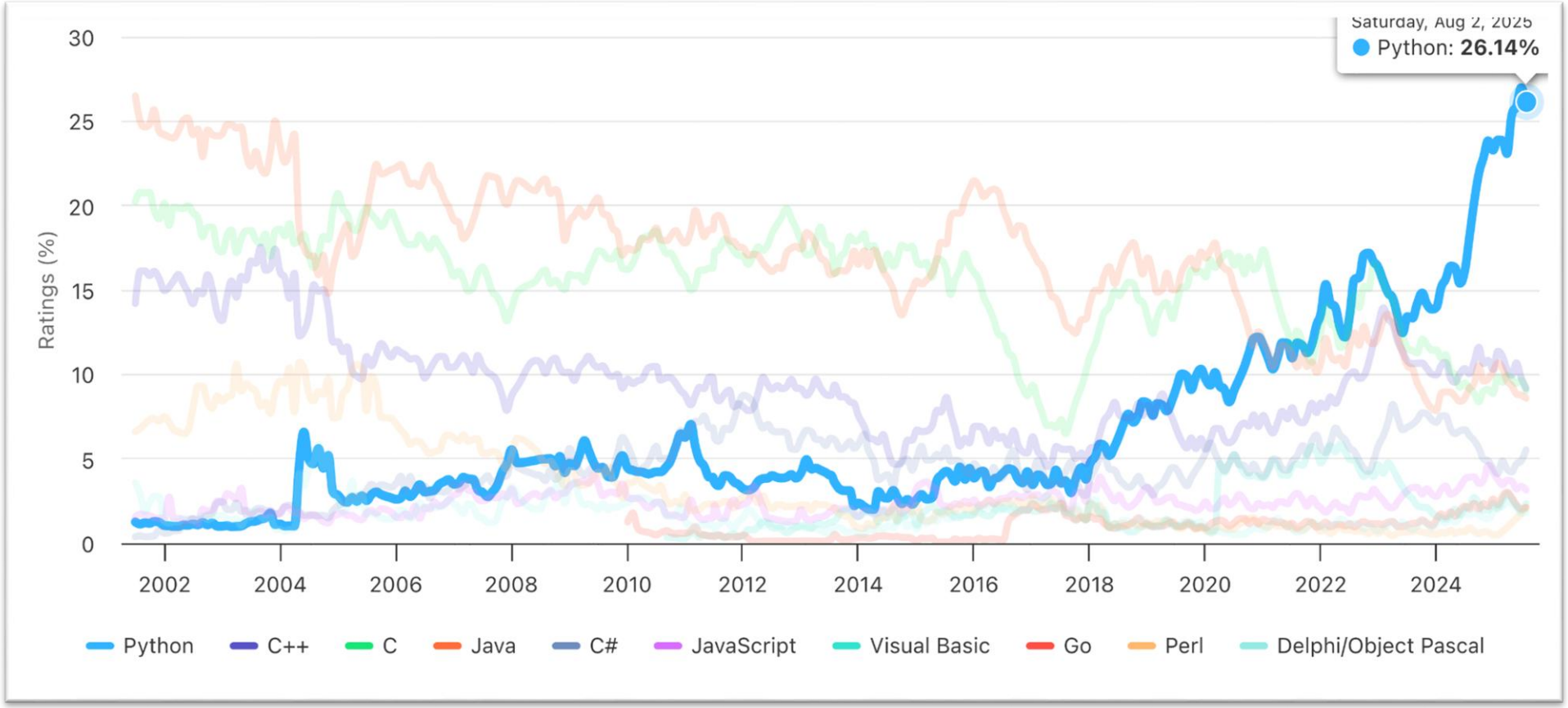
https://gitcode.com/cann/pyasc

PyAsc: PYthon for AScend C

CANN

Python成为智能时代的主流编程语言

□Python成为TIOBE历史上最受欢迎的编程语言（占比26.14%）



TIOBE编程语言排行榜（2025年8月）
Source： www.tiobe.com

https://gitcode.com/cann/pyasc

开发者的三个追问

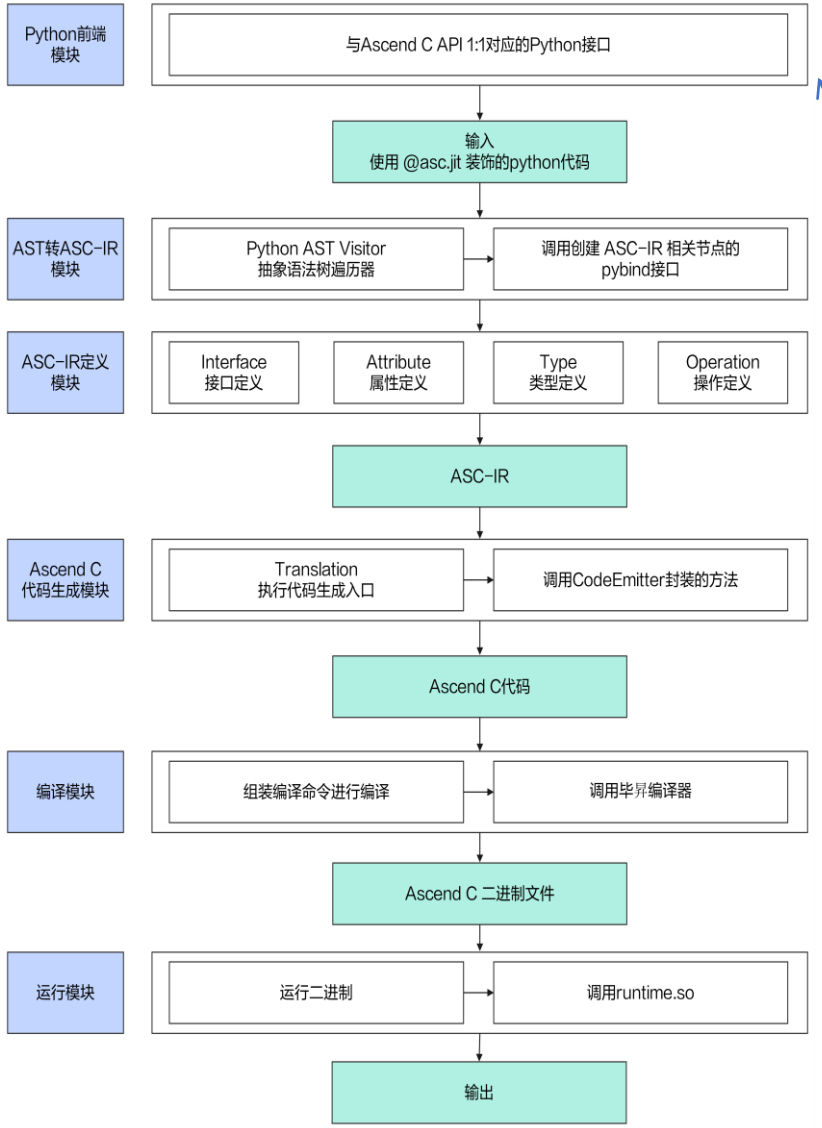
1. Python可以编写算子吗?
2. Python怎么编写高性能算子?
3. Python编写算子什么流程?

https://gitcode.com/cann/pyasc

PyAsc的作用——开发者视角和用户视角

开发者视角

PyAsc是一个提供与Ascend C的接口逐个对应的Python前端接口的项目，将Ascend C的算子开发范式迁移到Python语言中，拓展了昇腾计算生态



用户视角

PyAsc是一个可使用Python编程语法，可以利用现有Python计算生态，编写高效的昇腾NPU算子的开发语言，并可以使用Python的语言服务器，具有IDE的良好支持

https://gitcode.com/cann/pyasc

CANN

目录

CANN

PyAsc已支持API

□查阅PyAsc已支持的API

✓docs/python-api/language/index.md

asc.language.basic

Common operations

copy	**在 Vector Core 的不同内部存储单元（VECIN, VECCALC, VECOUT）之间进行数据搬运。**
data_cache_clean_and_invalid	用来刷新Cache，保证Cache与Global Memory之间的数据一致性。
data_copy	DataCopy系列接口提供全面的数据搬运功能，支持多种数据搬运场景，并可在搬运过程中实现随路格式转换和量化激活等操作。 该接口支持Local Memory与Global Memory之间的数据搬运，以及Local Memory内部的数据搬运。
data_copy_pad	DataCopyPad接口提供数据非对齐搬运的功能，其中从Global Memory搬运数据至Local Memory时，可以根据开发者的需要自行填充数据。
dump_tensor	基于算子工程开发的算子，可以使用该接口Dump指定Tensor的内容。
duplicate	将一个变量或立即数复制多次并填充到向量中。
get_block_idx	获取当前核的index，用于代码内部的多核逻辑控制及多核偏移量计算等。
get_block_num	获取当前任务配置的核数，用于代码内部的多核逻辑控制等。
get_data_block_size_in_bytes	获取当前芯片版本一个datablock的大小，单位为byte。 开发者可以根据datablock的大小来计算API指令中待传入的repeatTime、 DataBlock Stride、Repeat Stride等参数值。

https://gitcode.com/cann/pyasc

隶属Ascend C Sig：

- **https://gitcode.com/cann/pyasc**
- https://gitcode.com/cann/atvc
- https://gitcode.com/cann/asc-devkit
- https://gitcode.com/cann/asc-tools

asc.language.adv

Matmul

class **asc.language.adv.Matmul(a: MatmulType, b: MatmulType, c: MatmulType, bias: MatmulType | None = None, matmul_config: MatmulConfig | None = None, matmul_policy: MatmulPolicy | None = 0)**

class **asc.language.adv.Matmul(handle: Value)**

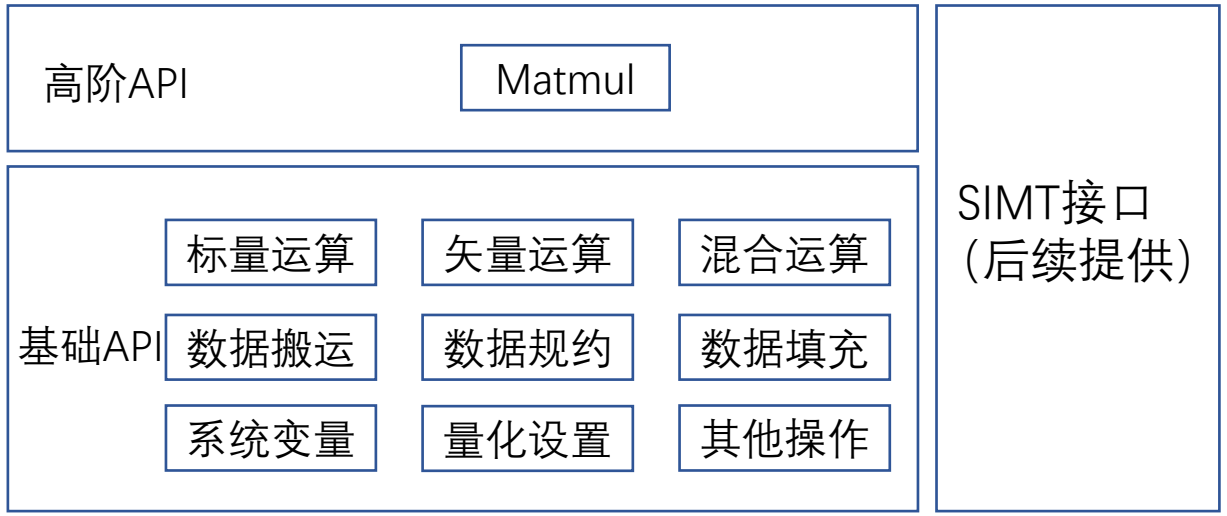
Ascend C提供一组Matmul高阶API，方便用户快速实现Matmul矩阵乘法的运算操作。 Matmul的计算公式为：C = A * B + Bias。

Matmul.async_get_tensor_c	**获取Iterate接口异步计算的结果矩阵。该接口功能已被GetTensorC覆盖，建议直接使用GetTensorC异步接口。**
Matmul.disable_bias	清除Bias标志位，表示Matmul计算时没有Bias参与。
Matmul.end	多个Matmul对象之间切换计算时，必须调用一次End函数，用于释放Matmul计算资源，防止多个Matmul对象的计算资源冲突。
Matmul.get_batch_tensor_c	调用一次get_batch_tensor_c，会获取C矩阵片，该接口可以与iterate_n_batch异步接口配合使用。 用于在调用iterate_n_batch迭代计算后，获取一片std::max(batch_a, batch_b) * singleCoreM * singleCoreN大小的矩阵分片。
Matmul.get_offset_c	预留接口，为后续功能做预留。 获取本次计算时当前分片在整个C矩阵中的位置。
Matmul.get_tensor_c	本接口和iterate接口配合使用，用于在调用iterate完成迭代计算后， 根据MatmulConfig参数中的ScheduleType取值获取一块或两块baseM * baseN大小的矩阵分片。
Matmul.init	灵活的自定义Matmul模板参数配置。
Matmul.iterate	每调用一次Iterate，会计算出一块baseM * baseN的C矩阵。
Matmul.iterate_all	调用一次iterate_all，会计算出singleCoreM * singleCoreN大小的C矩阵。

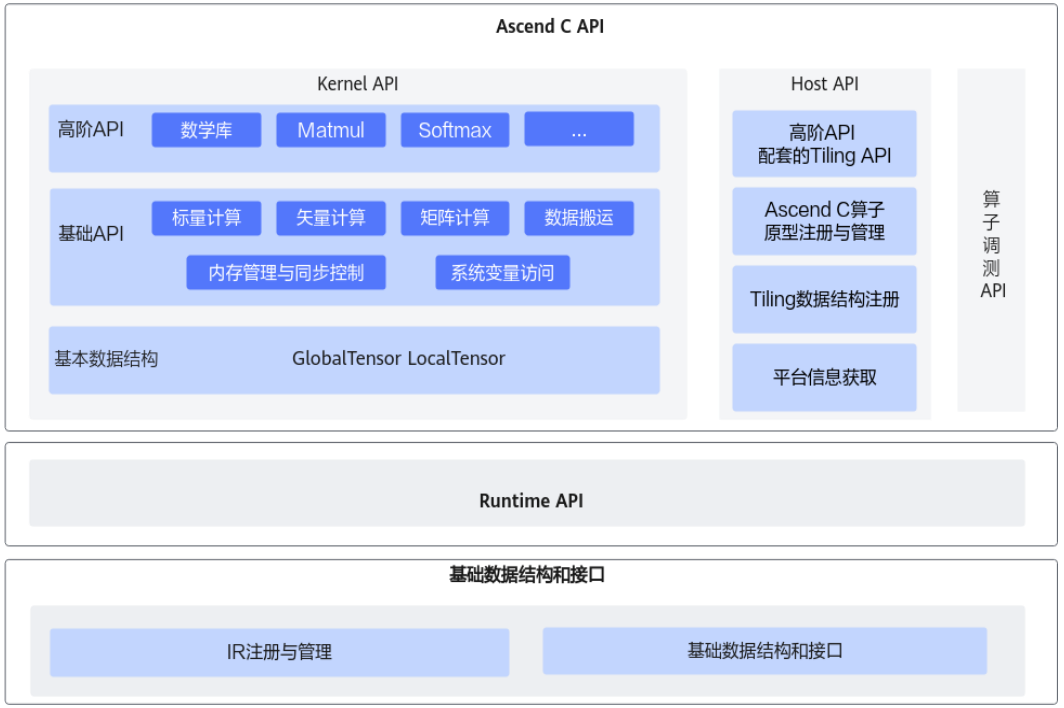
PyAsc已支持API

PyAsc API结构

- ✓PyAsc API按照运算类型进行归类，分阶段实现
- ✓目前已支持大部分基础API与Matmul相关API
- ✓提供了部分API实现算子的样例供开发者参考



Ascend C API结构

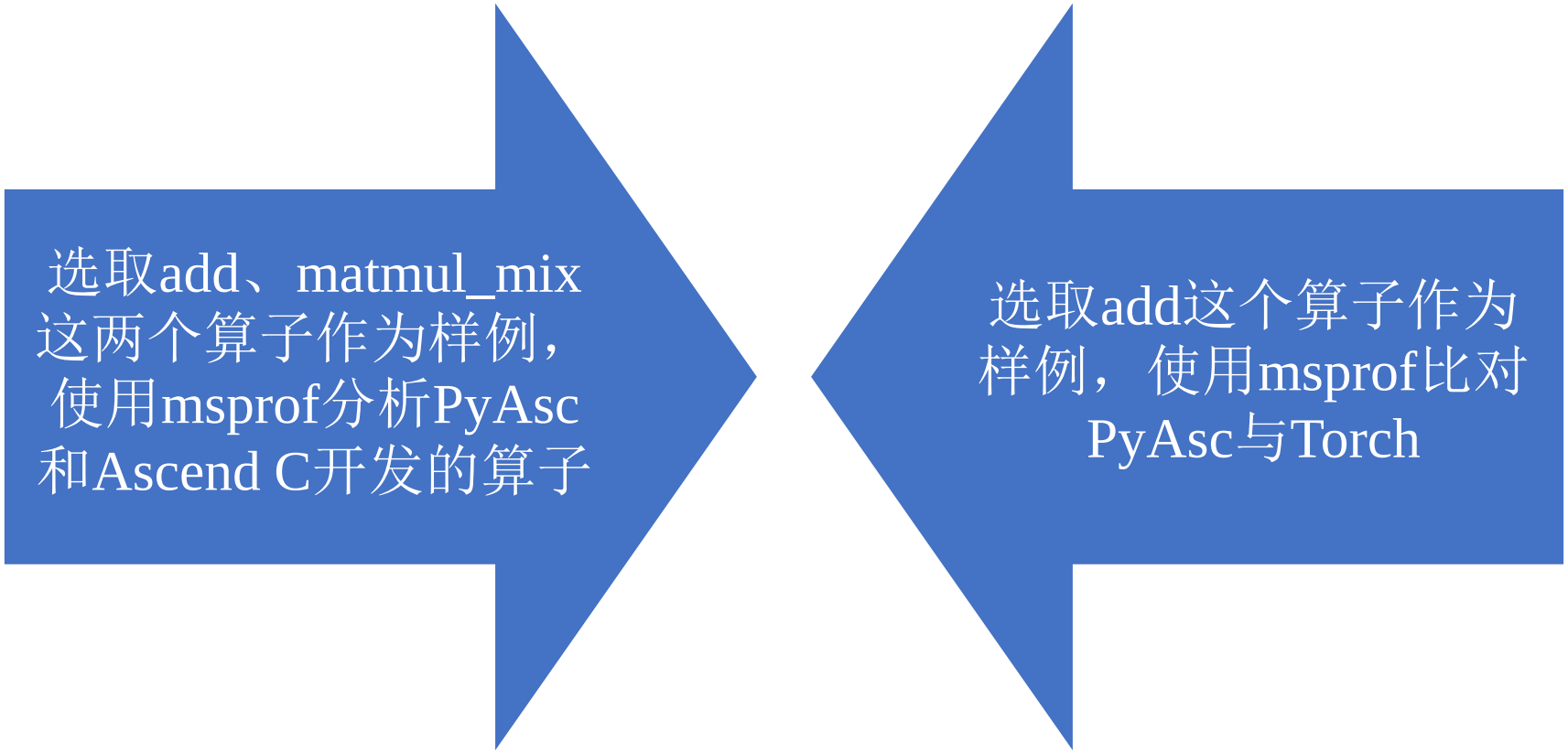


https://gitcode.com/cann/pyasc

目录

PyAsc性能分析

为了对pyasc的性能进行对照分析，选择样例进行了测试

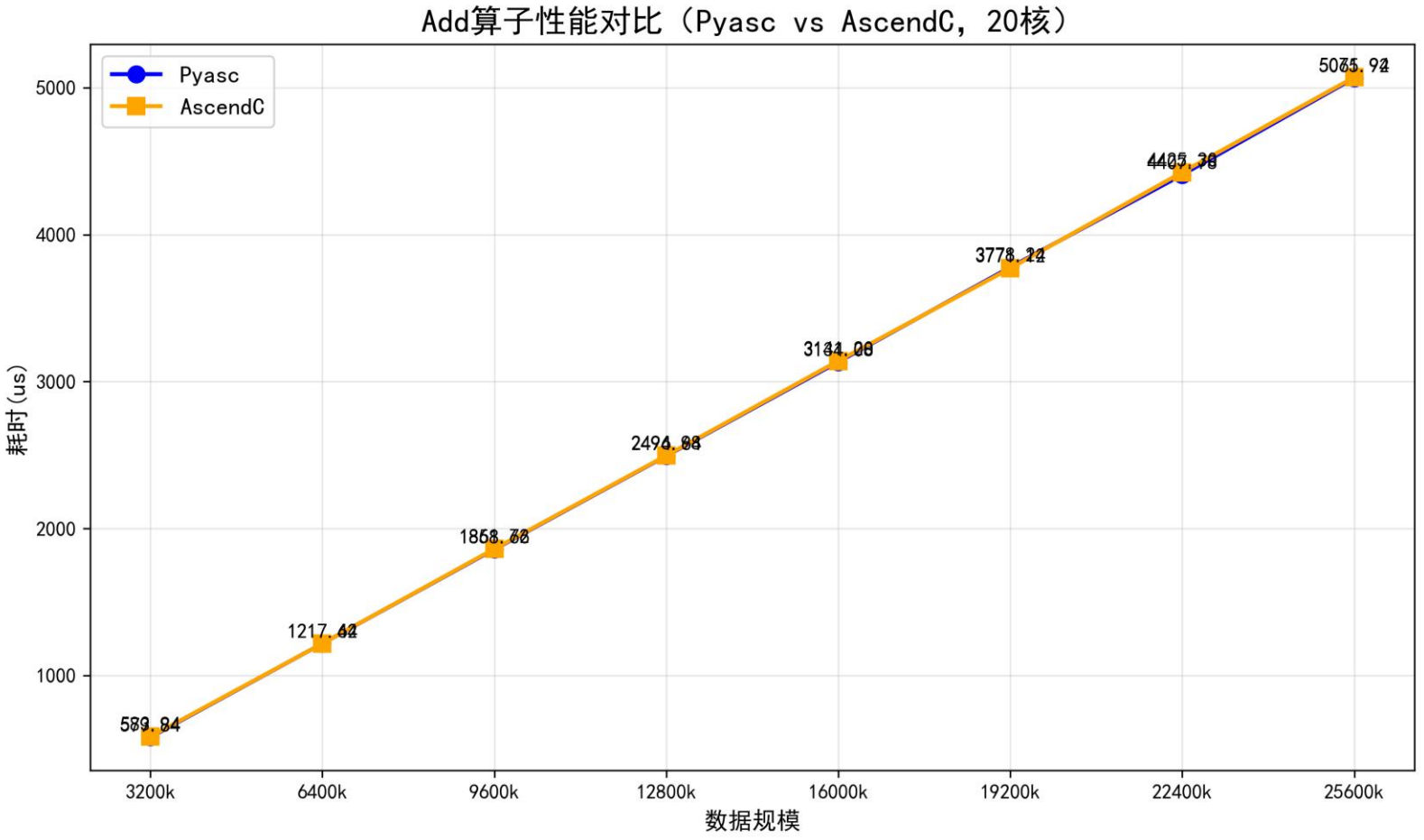


https://gitcode.com/cann/pyasc

PyAsc与Ascend C 对比：Add算子

数据规模（核数20）	PyAsc耗时(us)	Ascend C耗时(us)	相对耗时差比(公式:(PyAsc –Ascend C)/ Ascend C ×100%)
3200k	579.840	583.944	-0.70%
6400k	1217.420	1217.640	-0.02%
9600k	1858.660	1861.720	-0.16%
12800k	2494.640	2496.976	-0.09%
16000k	3134.080	3141.200	-0.23%
19200k	3778.240	3771.120	+0.19%
22400k	4407.784	4425.296	-0.40%
25600k	5065.920	5071.944	-0.12%

PyAsc与Ascend C 对比：Add算子

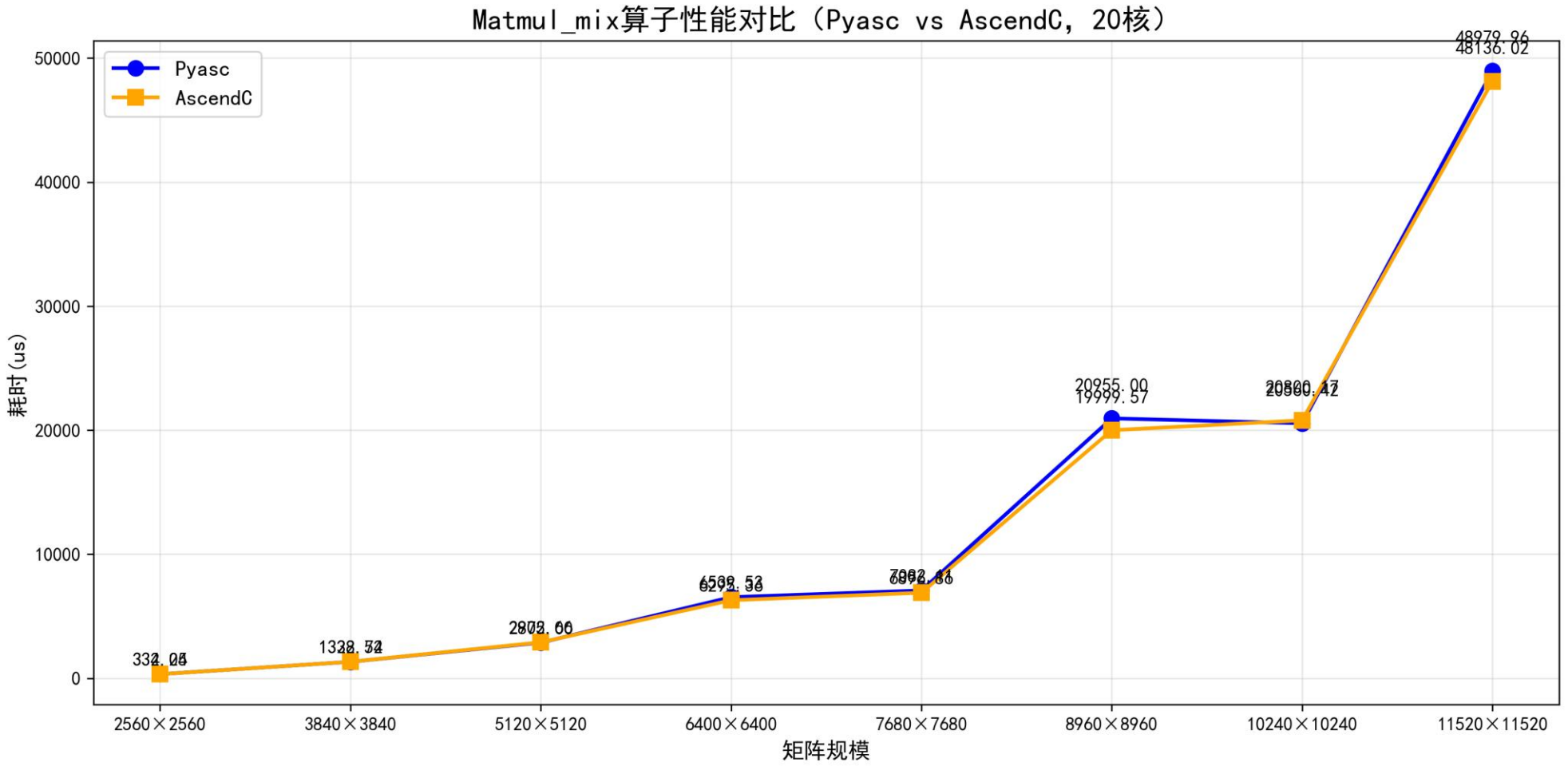


Add算子：
PyAsc和Ascend C性能基本持平

PyAsc与Ascend C 对比： Matmul_mix算子

矩阵规模（核数20）	PyAsc耗时(us)	Ascend C耗时(us)	相对耗时差比(公式:(PyAsc-Ascend C）/Ascend C ×100%)
2560×2560	334.25	332.04	+0.66%
3840×3840	1328.72	1332.54	-0.29%
5120×5120	2875.00	2902.66	-0.95%
6400×6400	6539.53	6295.36	+3.88%
7680×7680	7082.11	6896.86	+2.69%
8960×8960	20955.00	19999.57	+4.78%
10240×10240	20560.42	20800.17	-1.15%
11520×11520	48979.96	48136.02	+1.75%

PyAsc与Ascend C 对比： Matmul_mix算子



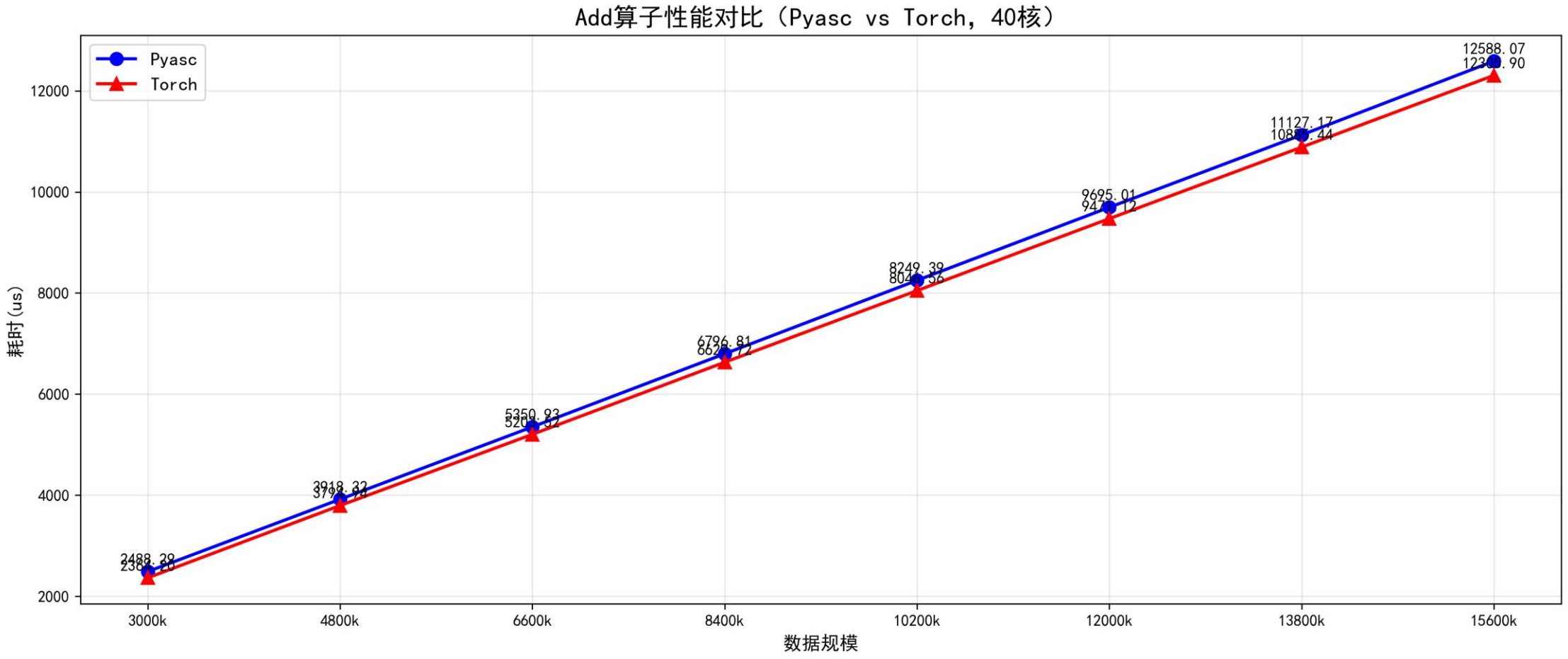
结论：Matmul_mix算子Pyasc和AscendC性能基本持平

CANN

PyAsc与Torch对比：Add算子

数据规模（核数40）	PyAsc耗时(us)	Torch耗时(us)	相对耗时差比(公式:(PyAsc - Torch)/Torch ×100%)
3000k	2488.29	2364.20	5.25%
4800k	3918.32	3794.94	3.25%
6600k	5350.93	5203.52	2.83%
8400k	6796.81	6629.72	2.52%
10200k	8249.39	8046.56	2.52%
12000k	9695.01	9471.12	2.36%
13800k	11127.17	10885.44	2.22%
15600k	12588.07	12305.90	2.29%

PyAsc与Torch对比：Add算子



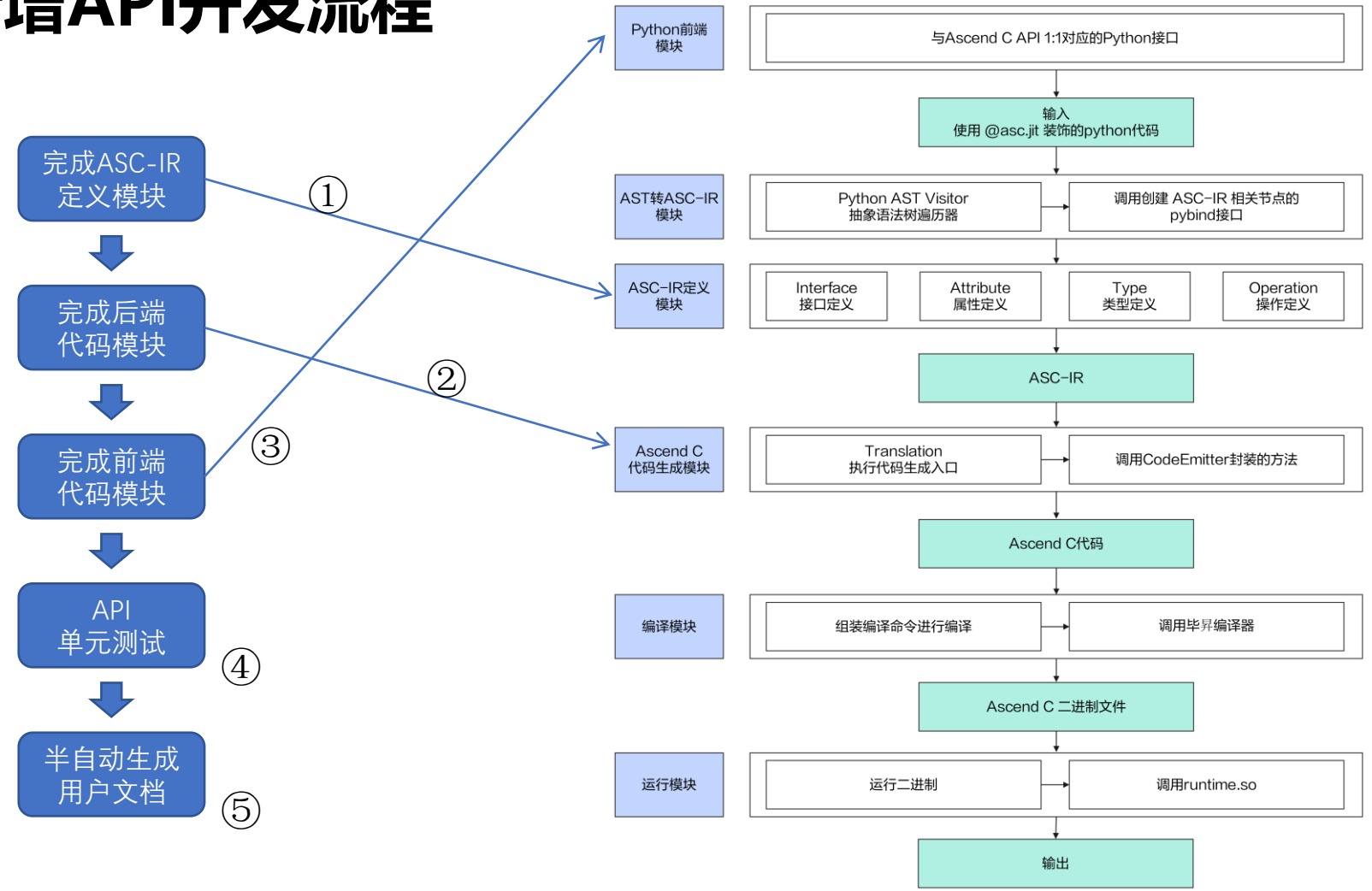
结论：Add算子PyAsc略劣于Torch。

CANN

目录

CANN

PyAsc新增API开发流程



下面，以Add算子为例

PyAsc新增API开发流程——1

实现ASC-IR定义模块：

在base.td里面已经预定义好BinaryOp等模板，若要新增开发的API符合模板，则直接使用模板进行定义，例如

```
defm Add : BinaryL023Op<"add", "Add", "operator+">;
```

若不符合模板，则可以基于APIOp等自定义开发

```
def AscendC_TQueBindAllocTensorOp : APIOp<"alloc_tensor", "AllocTensor"> {
  let summary = "Allocate tensor on queue wrapped buffer";
  let arguments = (ins AscendC_BaseQueueTypeInterface:$queue);
  let results = (outs AscendC_LocalTensor:$tensor);
  let assemblyFormat = [{
    $queue attr-dict `:` qualified(type($queue)) `,` qualified(type($tensor))
  }];
```

CANN

PyAsc新增API开发流程——2

后端代码自动生成

自动生成触发条件

当ASC-IR定义满足AscConstructor、AscMemberFunc或AscFunc特征时，genEmitter开关触发，自动启动后端代码生成流程，无需手动编写样板代码。

代码生成过程

框架依据assemblyFormat与参数顺序拼接Cpp片段，开发者仅需在 TD 文件中描述操作名、参数及结果，即可生成完整的Ascend C代码。

CANN

PyAsc新增API开发流程——2

若新增API不涉及数组类型数据，则可以在定义ASC-IR的时候增加[AscFunc]，自动生成Ascend C代码生成接口，否则需要定义开发后端代码生成的函数，如下

```
LogicalResult mlir::ascendc::printOperation(CodeEmitter& emitter, ascendc::GlobalTensorGetPhyAddrOp op)
{
    FAIL_OR(emitter.emitVariableDeclaration(op->getResult(0), false));
    auto& os = emitter.ostream();
    os << " = " << emitter.getOrCreateName(op.getTensor()) << "." << op.getAPIName() << "(";
    if (auto offset = op.getOffset()) {
        os << emitter.getOrCreateName(offset);
    }
    os << ")";
    return success();
}
```

CANN

PyAsc新增API开发流程——3

编写Python函数重载

```
@overload
def add(dst: LocalTensor, src0: LocalTensor, src1: LocalTensor, count: int) -> None:
    ...

@overload
def add(dst: LocalTensor, src0: LocalTensor, src1: LocalTensor, mask: int, repeat_times: int,
        repeat_params: BinaryRepeatParams) -> None:
    ...


@overload
def add(dst: LocalTensor, src0: LocalTensor, src1: LocalTensor, mask: List[int], repeat_times: int,
        repeat_params: BinaryRepeatParams) -> None:
    ...
```

CANN

PyAsc新增API开发流程——3

实现Python主函数

```
@require_jit
@set_docstring("sum", cpp_name="Add")
def add(dst: LocalTensor, src0: LocalTensor, src1: LocalTensor, *args, **kwargs) -> None:
    # 按照Ascend C的函数定义入参需要支持位置参数args，即不需要显示声明传入的参数
    builder = global_builder.get_ir_builder()
        # 全局管理器，用于生成ast节点以及生成ir
    op_impl("add", dst, src0, src1, args, kwargs,
            builder.create_asc_AddL0Op,
            builder.create_asc_AddL2Op)
    # builder.create_asc_xxxOp
    # 代表对应的ir创建方法，需要确保ir处定义了对应op，并且参数对应
    # build_l0 和 build_l2 对应双目指令不带mask和带mask两种计算模式
```

CANN

PyAsc新增API开发流程——4

Python UT验证流程与案例

本模块基于pytest框架构建，整体实现过程如下：
a. 初始化环境，注入mock_launcher_run桩函数；
b. 定义内核函数，调用被测对象及方法；触发内核运行；
c. 通过断言验证编译以及执行流程是否按照预期触发。

以add为例，在python\test\unit\language\basic\test_vector_binary.py文件中编写UT测试代码，设置PYASC_DUMP_PATH环境变量后，运行pytest ./python/test/unit/language/basic/test_vector_binary.py即可完成测试，并能够在PYASC_DUMP_PATH指定的目录查看生成的MLIR和Ascend C代码。

CANN

PyAsc新增API开发流程——4

```
def test_add_kernel(mock_launcher_run):

    @asc.jit
    def add_kernel():
        x_local = asc.LocalTensor(dtype=asc.float16, pos=asc.TPosition.VECIN,
                                  addr=0, tile_size=512)
        y_local = asc.LocalTensor(dtype=asc.float16, pos=asc.TPosition.VECIN,
                                  addr=0, tile_size=512)
        z_local = asc.LocalTensor(dtype=asc.float16, pos=asc.TPosition.VECOUT,
                                  addr=0, tile_size=512)
        asc.add(z_local, x_local, y_local, count=512)
        params = asc.BinaryRepeatParams(1, 1, 1, 8, 8, 8)
        asc.add(z_local, x_local, y_local, mask=512,
                repeat_times=1, repeat_params=params)
        uint64_max = 2**64 - 1
        mask = [uint64_max, uint64_max]
        asc.add(z_local, x_local, y_local, mask=mask,
                repeat_times=1, repeat_params=params)


    add_kernel[1]()
    assert mock_launcher_run.call_count == 1
```

CANN

PyAsc新增API开发流程——5

本项目采用Sphinx工具，通过提取项目 Python 代码（如模块、类、函数）中的 docstring（文档字符串）自动生成标准化的 Python API 文档。具体步骤如下：

1. 编写docstring内容：确保修改后的函数、类、模块已按规范编写 docstring

2. 进入docs目录并重新生成文档：在 docs 目录下执行如下文档生成命令，Sphinx 将自动识别代码变更并更新文档内容。

```
# Linux/Mac 环境
cd docs && make markdown
# Windows 环境
cd docs && sphinx-build -b markdown . /_build/markdown
```

3. 验证文档更新结果：重新打开docs/_build/markdown/index.md，导航到该修改的模块、函数，确认文档内容已同步更新。

CANN

PyAsc新增枚举类型开发流程

枚举类型的开发和遇到的问题、解决的方案（以CacheLine为例）

1. 遇到的问题

- Python 枚举传不上 IR（缺少 pybind/symbolize）
- ascendc.cpp 打印错误（TD 与 Python 值不一致）

2. 解决方案

- TD / Python / C++ 三处保持同名同值
- Python 统一使用 IntEnum
- pybind 必须绑定并提供 symbolize()

3. 开发步骤（TD → Python → C++ 保持一致）

- 完成ASC-IR定义模块: def AscendC_CacheLineAttr : I32EnumAttr<"CacheLine", "", [I32EnumAttrCase("SINGLE_CACHE_LINE", 0,"single_cache_line")]
- 完成前端Cache代码模块：Line(IntEnum): SINGLE_CACHE_LINE = 0
- 自动完成绑定功能：.value("SINGLE_CACHE_LINE", ascendc::CacheLine::SINGLE_CACHE_LINE)

```
. def_static("symbolize", [ ](uint8_t v) -> ascendc::CacheLine { return static_cast<ascendc::CacheLine>(v); });
```

CANN

PyAsc新增API开发流程

Python与Ascend C 的语法差异与解决方案

模板参数: Python 无模板 -> 映射规则：模板转运行时参数

函数重载: Python 不支持 -> 使用 @overload 声明重载签名，@require_jit 实现分发

默认参数: 保持 Ascend C 中的默认值作为 Python 默认值

引用: 通过 IRHandle/对象封装（to_ir/from_ir）传递

CANN

调用新增Add接口编写算子

- 用户可以直接使用python/tutorials/01_add/add.py提供的代码，验证新增的Add接口
  - ✓其中，使用Add接口定义开发了一个用于进行向量计算的核函数

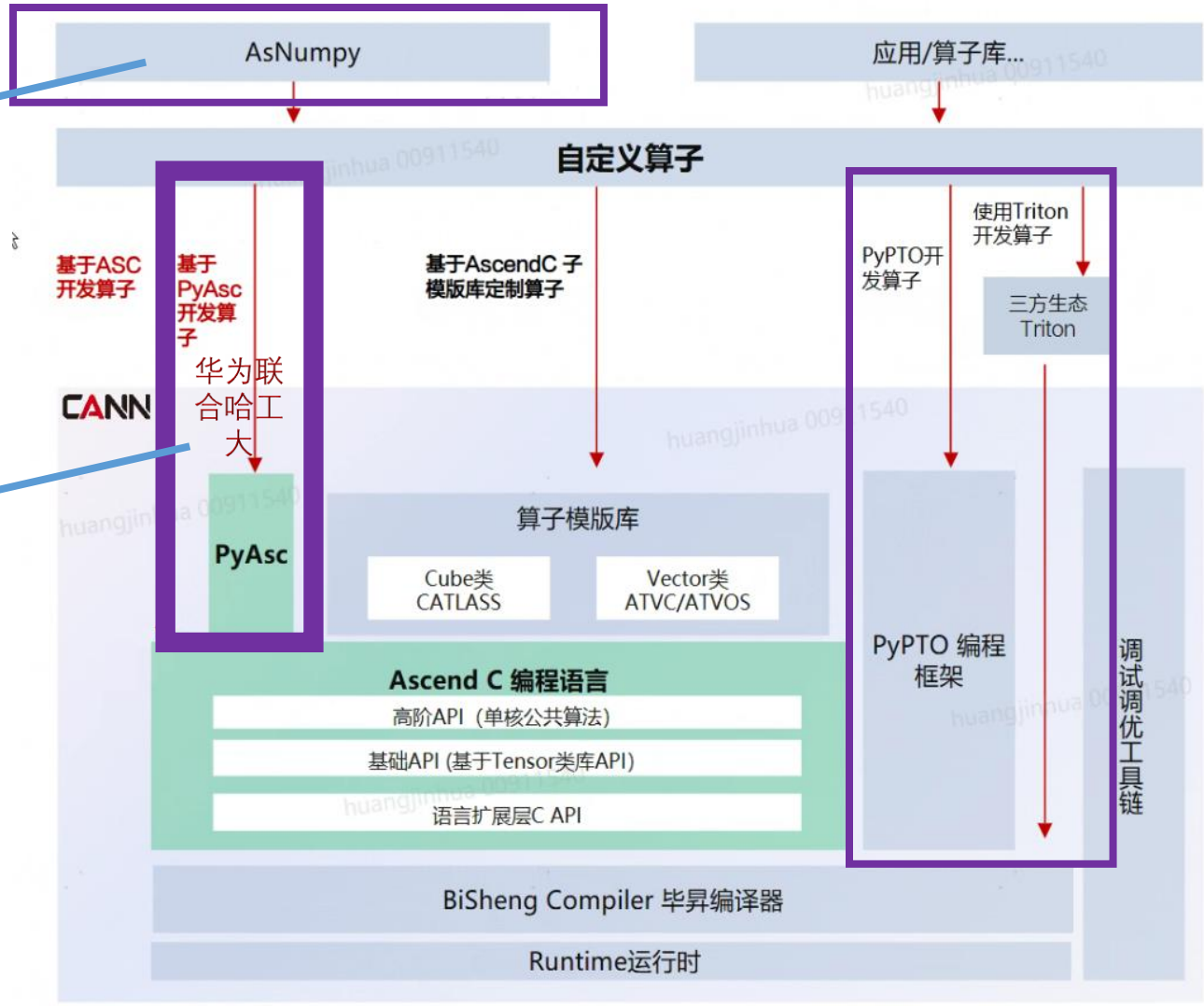
```python
@asc.jit
def vadd_kernel(x: asc.GlobalAddress, y: asc.GlobalAddress, z: asc.GlobalAddress, block_length: int):

    offset = asc.get_block_idx() * block_length
    x_gm = asc.GlobalTensor()
    y_gm = asc.GlobalTensor()
    z_gm = asc.GlobalTensor()
    x_gm.set_global_buffer(x + offset, block_length)
    y_gm.set_global_buffer(y + offset, block_length)
    z_gm.set_global_buffer(z + offset, block_length)

    tile_length = block_length // TILE_NUM // BUFFER_NUM

    data_type = x.dtype
    buffer_size = tile_length * BUFFER_NUM * data_type.sizeof()

    # Init a Tensor based on the specified logical position/address/length
    x_local = asc.LocalTensor(data_type, asc.TPosition.VECIN, 0, tile_length * BUFFER_NUM)
    y_local = asc.LocalTensor(data_type, asc.TPosition.VECIN, buffer_size, tile_length * BUFFER_NUM)
    z_local = asc.LocalTensor(data_type, asc.TPosition.VECOUT, buffer_size + buffer_size, tile_length * BUFFER_NUM)

    for i in range(TILE_NUM * BUFFER_NUM):
        buf_id = i % BUFFER_NUM

        # Operator[] return a new LocalTensor/GlobalTensor with offset from the original starting address
        asc.data_copy(x_local[buf_id * tile_length:], x_gm[i * tile_length:], tile_length)
        asc.data_copy(y_local[buf_id * tile_length:], y_gm[i * tile_length:], tile_length)

        # Synchronization instructions between different pipelines in the same core
        asc.set_flag(asc.HardEvent.MTE2_V, buf_id)
        asc.wait_flag(asc.HardEvent.MTE2_V, buf_id)

        asc.add(z_local[buf_id * tile_length:], x_local[buf_id * tile_length:], y_local[buf_id * tile_length:],
                tile_length)

        asc.set_flag(asc.HardEvent.V_MTE3, buf_id)
        asc.wait_flag(asc.HardEvent.V_MTE3, buf_id)

        asc.data_copy(z_gm[i * tile_length:], z_local[buf_id * tile_length:], tile_length)

        asc.set_flag(asc.HardEvent.MTE3_MTE2, buf_id)
        asc.wait_flag(asc.HardEvent.MTE3_MTE2, buf_id)
```

https://gitcode.com/cann/pyasc

小结：算子的Python编程生态

哈工大联合华为

华为联合哈工大



https://gitcode.com/cann/pyasc

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态

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

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

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

上CANN社区获取干货

关注CANN公众号获取资讯

CANN