Ascend C: 构建多级API，支撑多维场景算子开发

作者：黄金华

时间：2025/12/01

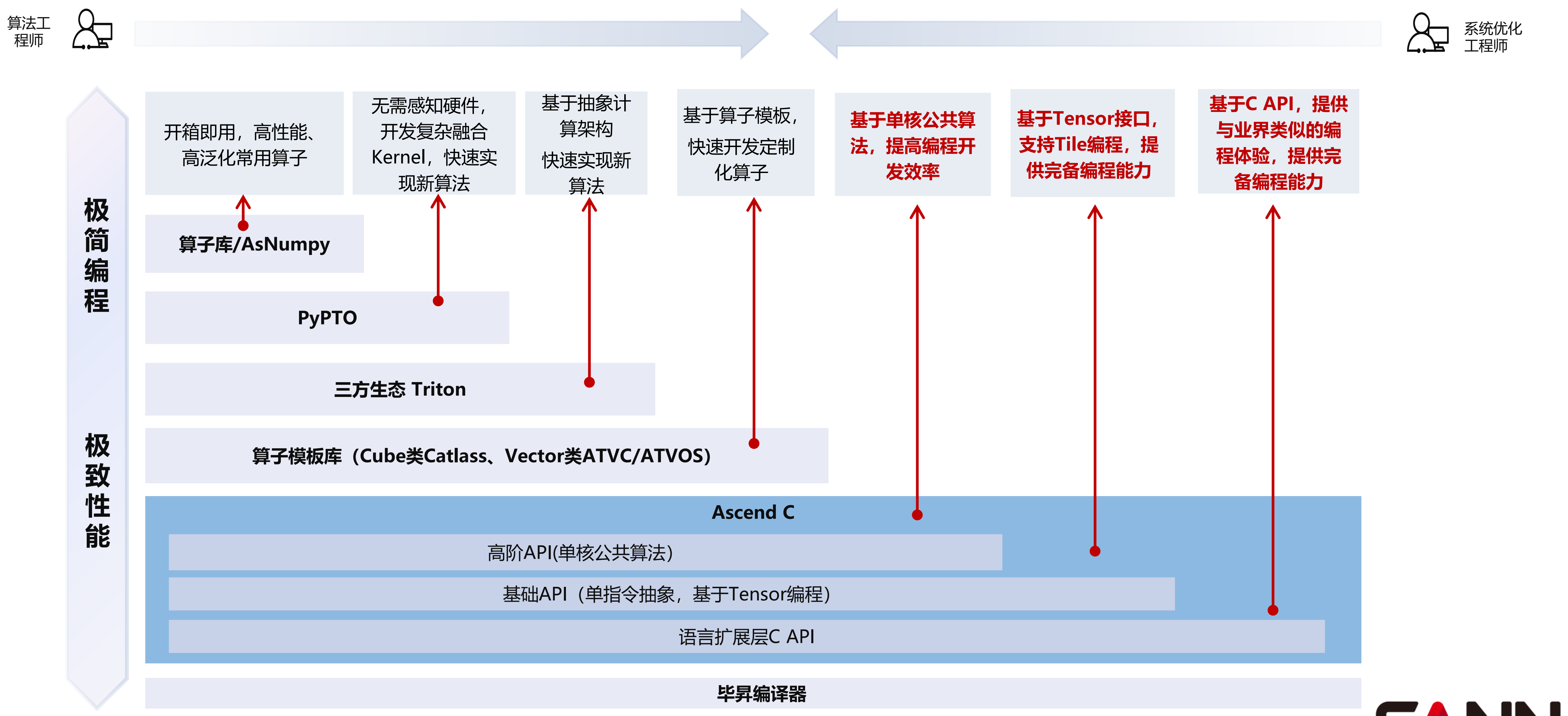
CANN

目录

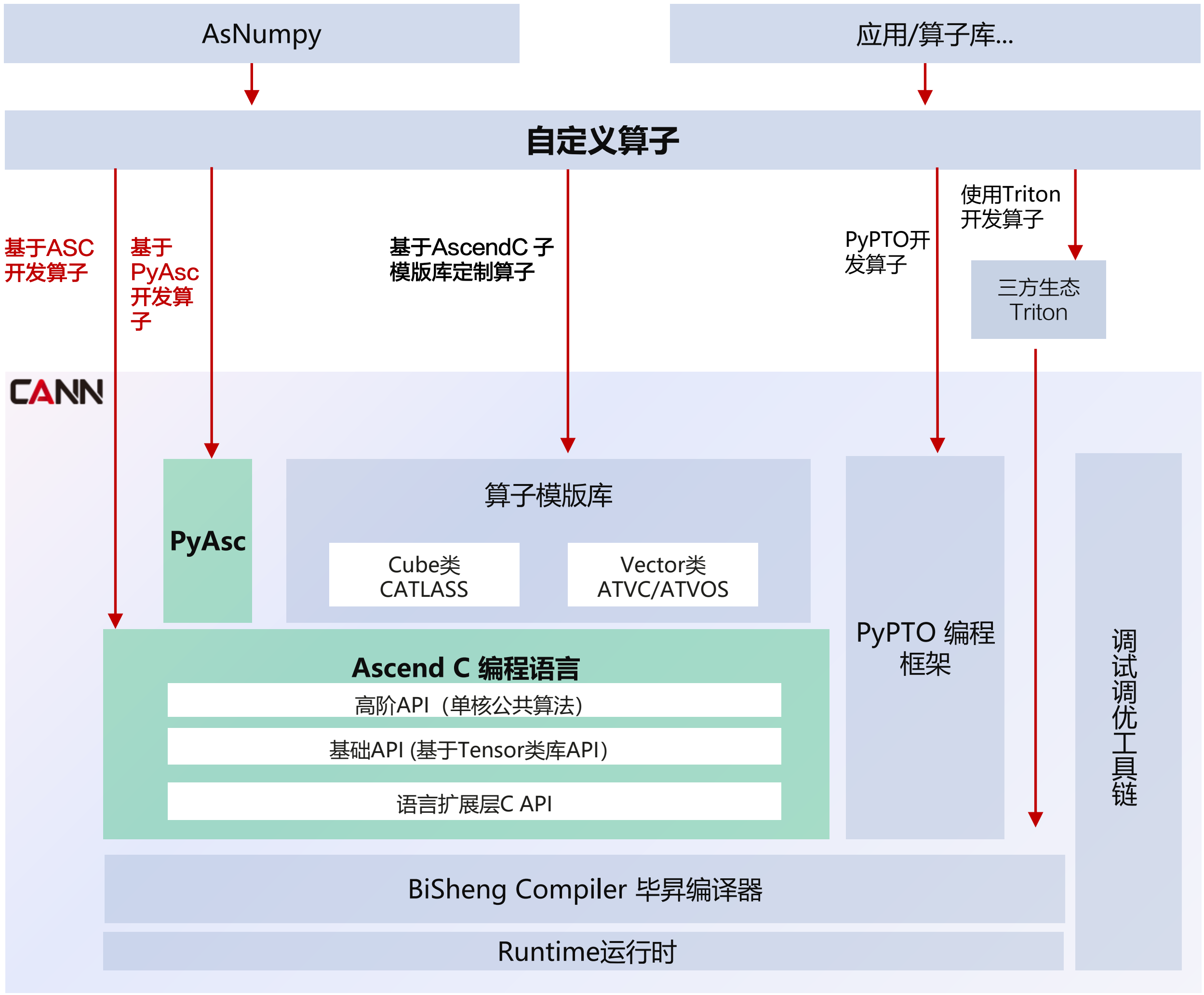
https://gitcode.com/cann/asc-devkit
https://gitcode.com/cann/asc-tools

CANN

Ascend C：能基于“手工”支撑实现极致性能



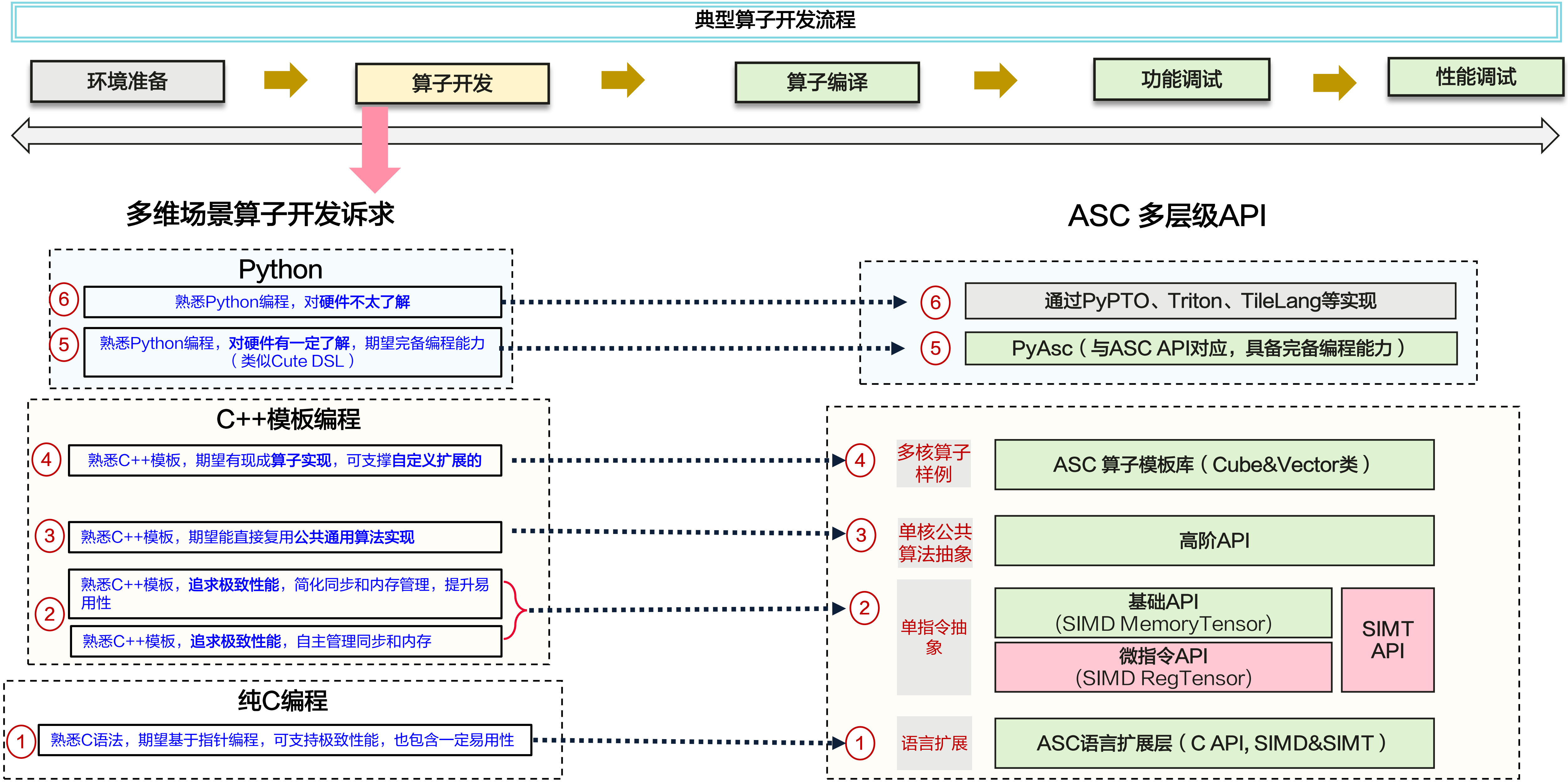
Ascend C: 基于C/C++扩展的算子编程语言



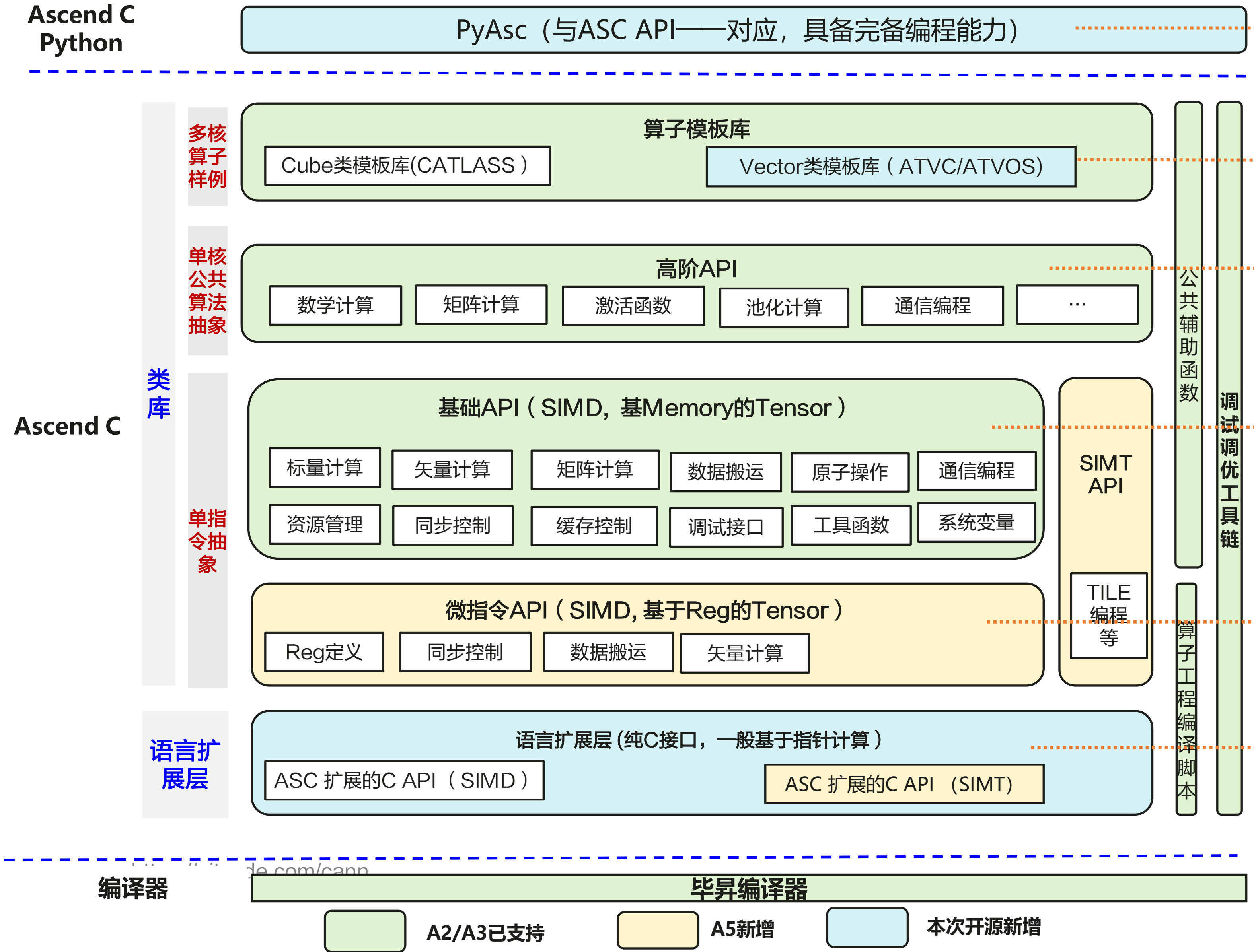
- **基于C/C++扩展，遵循C/C++标准规范；**
- **提供底层芯片完备编程能力，支撑实现极致性能；**
- **构建多级接口，满足多场景算子开发诉求，匹配业界开发习惯；**
- **支持异构编程和<<<>>>直调**
- **一套代码支持CPU/NPU孪生调试**

https://gitcode.com/cann/asc-devkit
https://gitcode.com/cann/asc-tools

Ascend C算子编程：构建多层级API，支撑多维场景算子开发诉求



Ascend C算子编程架构：构建多层级API，支撑多维场景算子开发诉求



Python：支持与ASC API一一对应能力，提供**Python完备编程能力**，完善Python编程生态

C++类库：基于**模板提供算子完整实现**，简化Tiling开发，**支撑用户自定义修改**

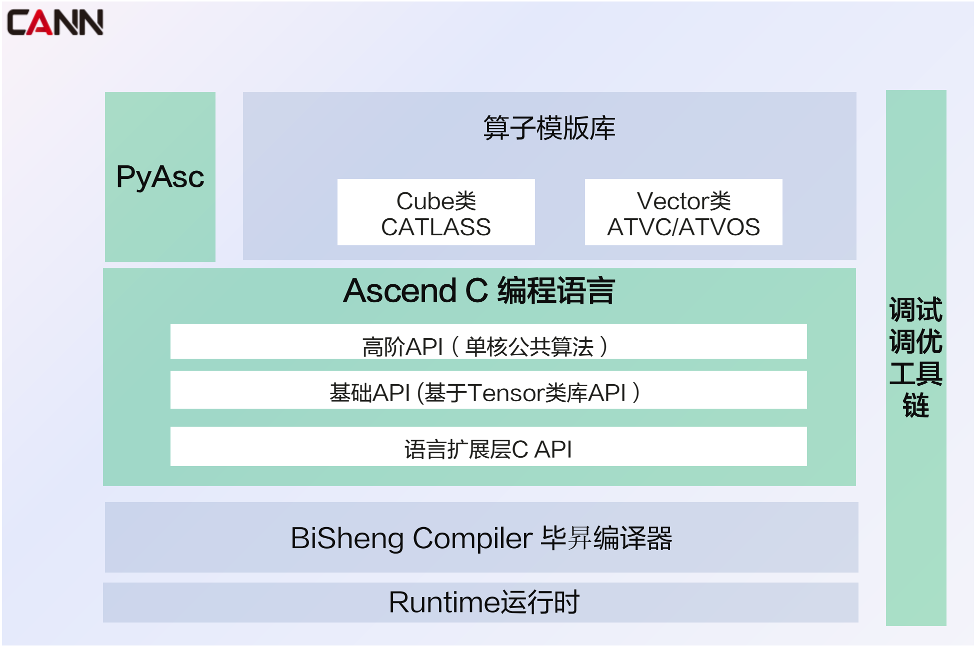
C++类库：提供**单核公共算法**的实现，简化单核算法实现

C++类库：**基于Tensor接口**，支持Cube和Vector编程，支撑芯片**完备编程能力**；Tensor新增支持Layout，实现Tile编程能力，提升易用性

C++类库：A5新增，支持**寄存器Tensor**完备编程能力，主要用于**向量计算**

C语言：**提供语言扩展层C API**，支持与业界一致的SIMD/SIMT编程体验(A5新增SIMT、SIMT&SIMD混编能力）

Ascend C 分仓分包方案：支持按需安装，独立升级



No	仓	说明
1	CANN/**asc-devkit**	● ASC 算子开发的需要的**最小集**（含API、编译脚本）；
2	CANN/**asc-tools**	● Ascend C 算子调试工具，包括CPU孪生调试等
3	CANN/pyasc	● Ascend C Python前端，与基础API/高阶API一一对应，完善Python编程生态；
4	CANN/atvos	● Vector类算子模板库
5	CANN/atvc	● Vector类Compute Only 模板库

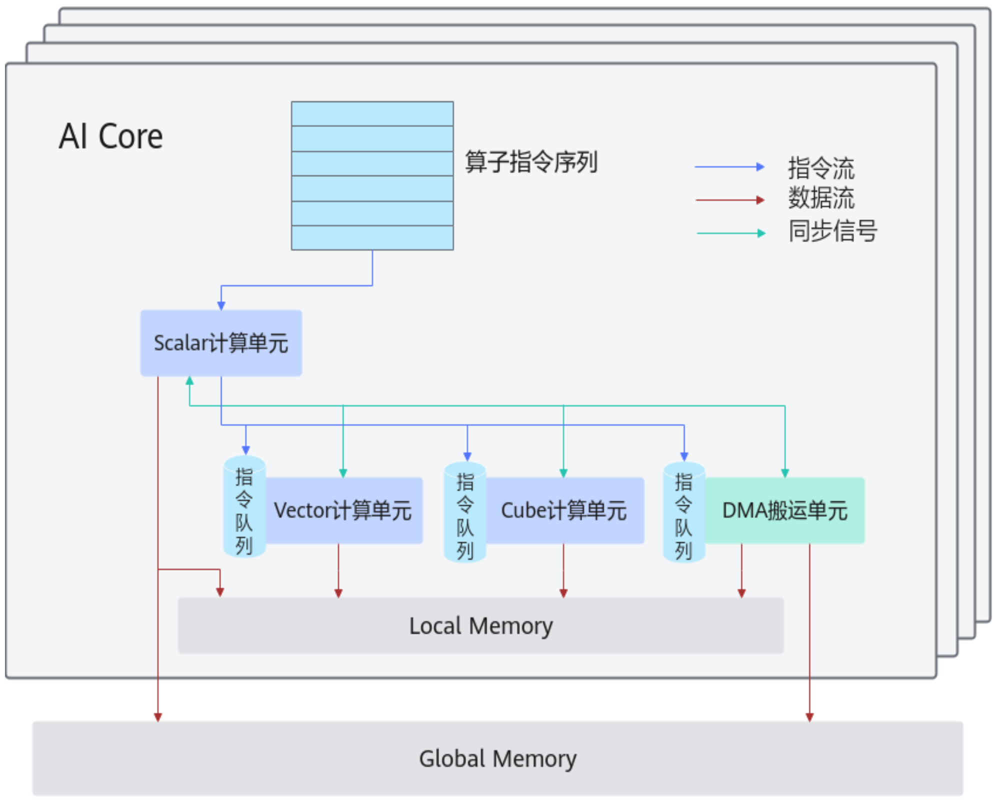
https://gitcode.com/cann/asc-devkit
https://gitcode.com/cann/asc-tools

CANN

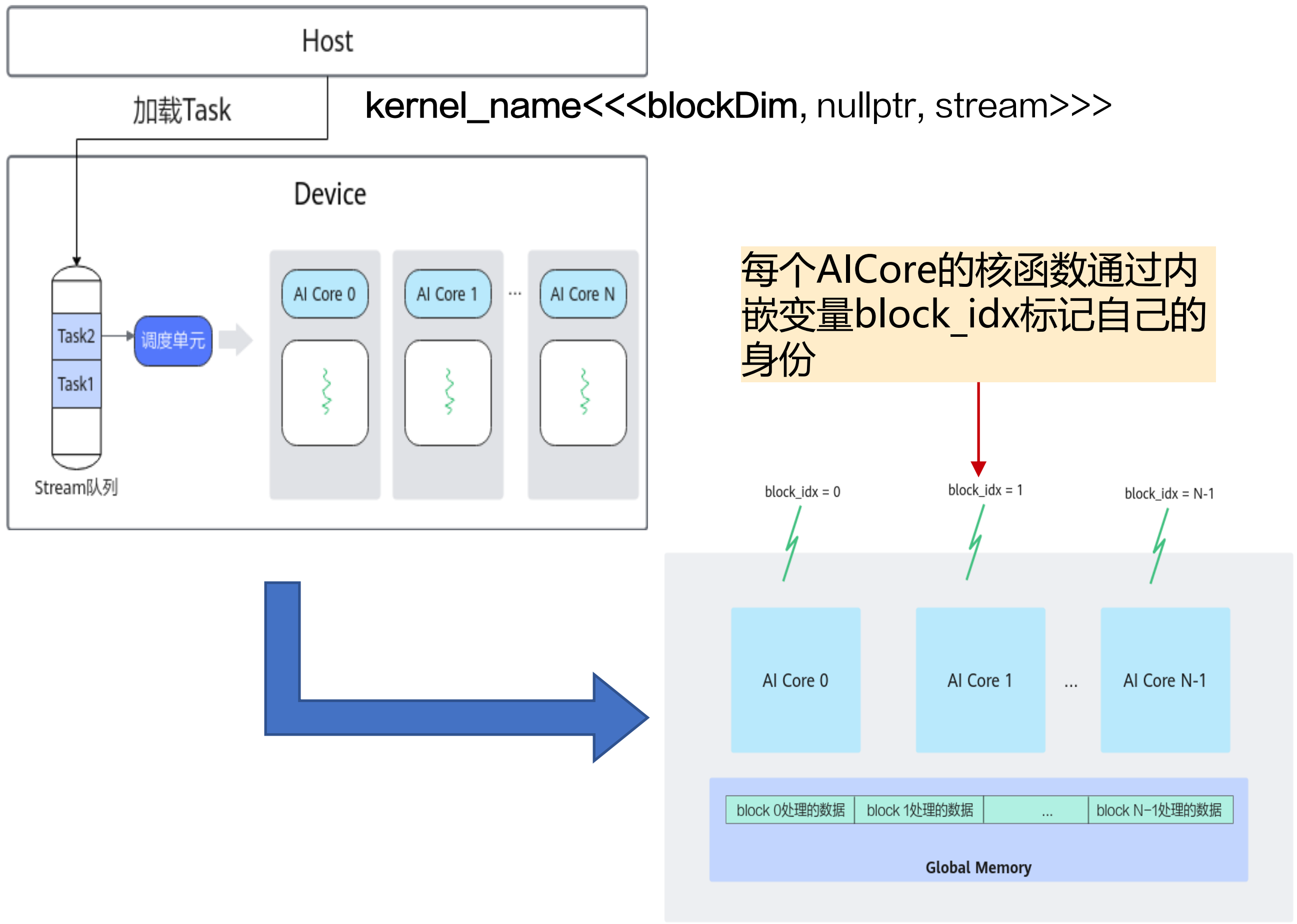
目录

Ascend C 编程模型简介

NPU抽象硬件架构



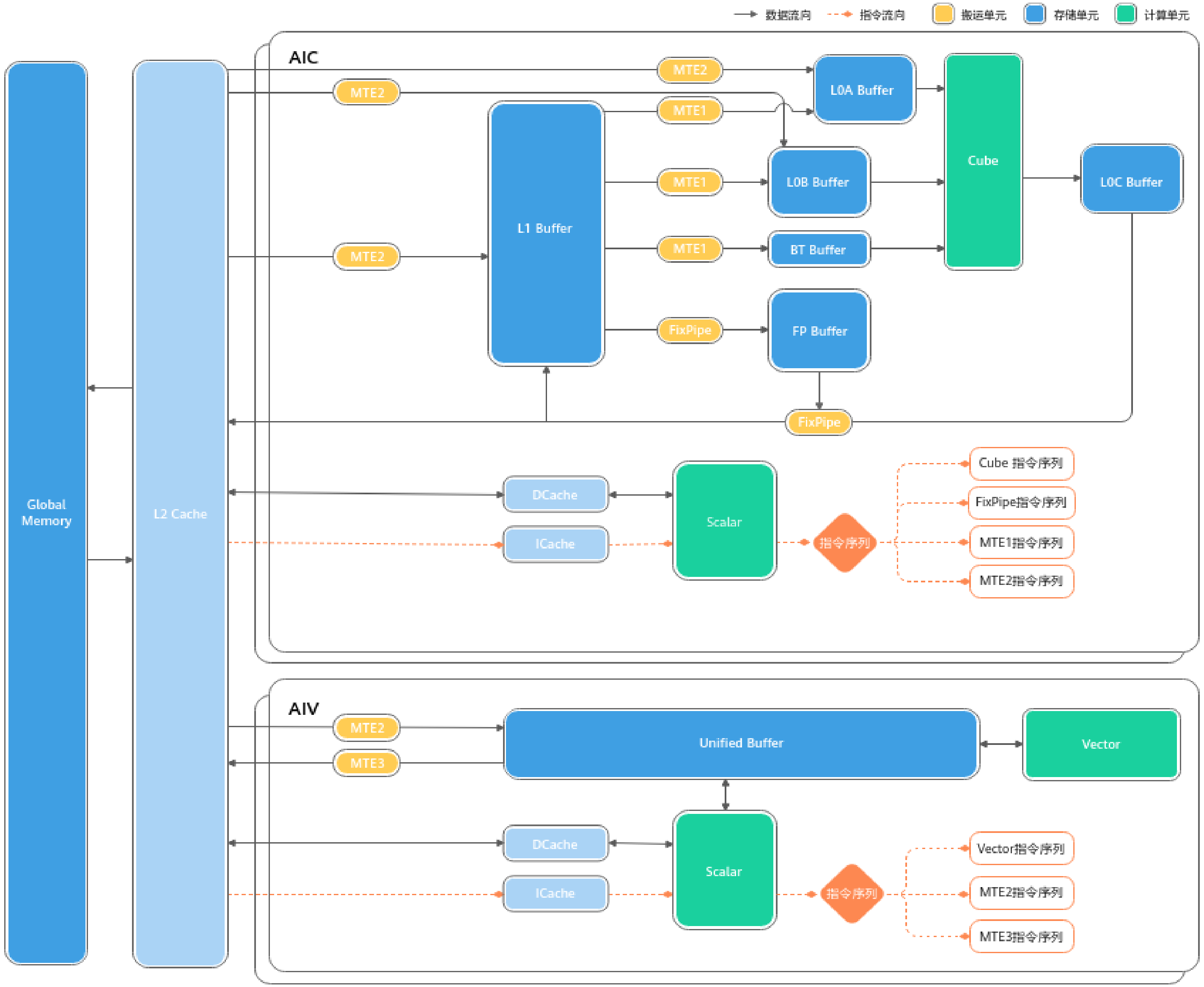
Ascend C SPMD 编程模型 (SIMD)



https://gitcode.com/cann/asc-devkit
https://gitcode.com/cann/asc-tools

CANN

A2/A3 硬件架构介绍

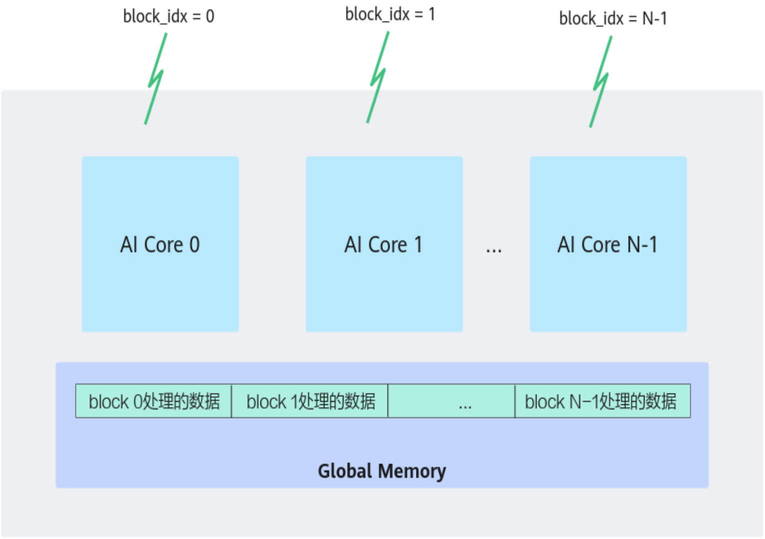


•**计算单元**：包括Cube（矩阵）计算单元、Vector（矢量）计算单元和Scalar（标量）计算单元

•**存储单元**：包括L1 Buffer、L0A Buffer、L0B Buffer、L0C Buffer、Unified Buffer、BiasTable Buffer、Fixpipe Buffer等专为高效计算设计的存储单元。

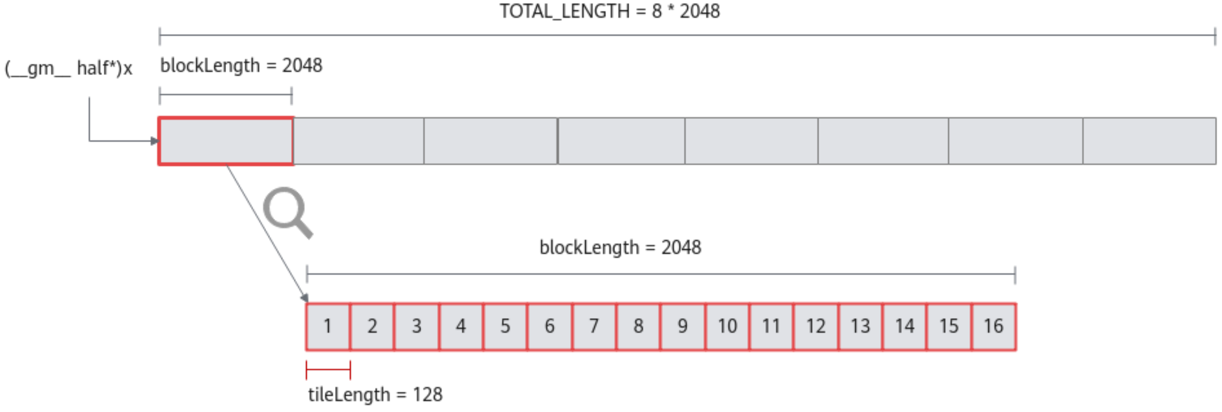
•**搬运单元**：包括MTE1、MTE2、MTE3和FixPipe，用于数据在不同存储单元之间的高效传输。

Ascend C编程范式(SIMD)：以Add为例介绍数据切分策略



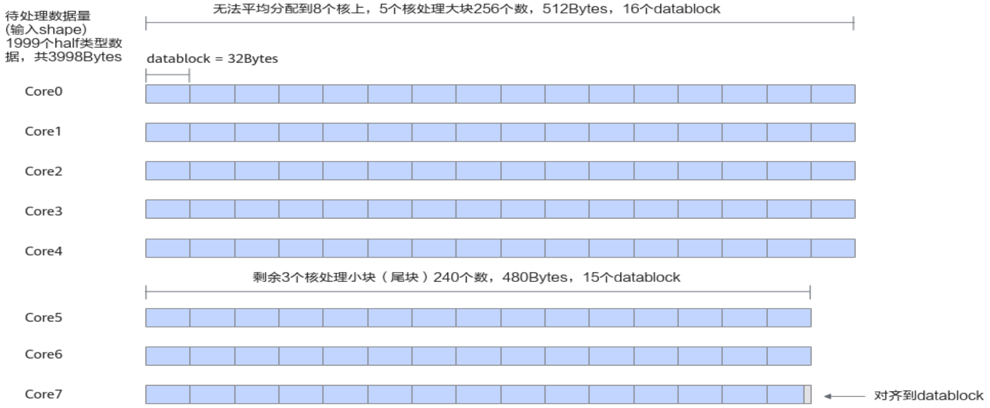
- **全对齐场景**

① 总数据长度8*2048
② 每个核处理2048个数据，每128个数据是基本块，总共16次



- **未对齐场景**

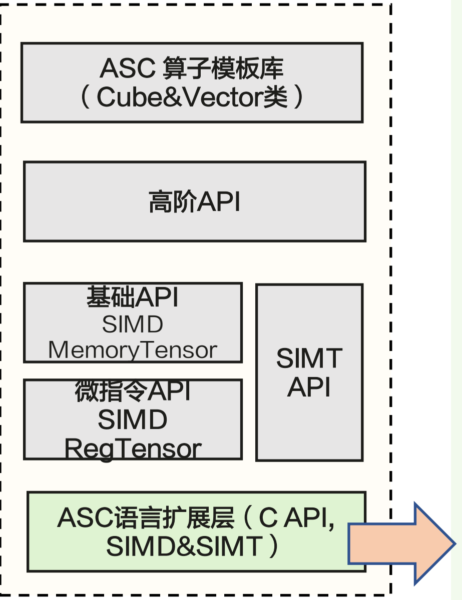
① **尾块**：数据能平分到每个核，但核内数据无法均分（如上图：blockLength=2047）
② **尾核**：数据不能平分到每个核，部分核处理数据较少；



https://gitcode.com/cann/asc-devkit
https://gitcode.com/cann/asc-tools

Ascend C编程范式(1): 基于ASC 语言扩展层C API的Add算子示例

基于语言扩展层接口C api，自主管理内存和同步，提供业界通用编程体验



示例一：基于同步搬运/计算接口

```
#define TILE_LENGTH_PER_CORE  2048

__global__ __aicore__ void add (__gm__ float* x,
__gm__ float* y, __gm__ float* z)
{
    KERNEL_TASK_TYPE(KERNEL_TYPE_AIV_ONLY);
    asc_init ();

    __ubuf__ float xIn[TILE_LENGTH_PER_CORE ];
    __ubuf__ float yIn[TILE_LENGTH_PER_CORE ];
    __ubuf__ float zOut[TILE_LENGTH_PER_CORE ];

     asc_copy_gm2ub_sync(xIn, x + block_idx *
TILE_LENGTH_PER_CORE, TILE_LENGTH_PER_CORE);
     asc_copy_gm2ub_sync(yIn, y + block_idx *
TILE_LENGTH_PER_CORE, TILE_LENGTH_PER_CORE);

     asc_add_sync(zOut, xIn, yIn,
TILE_LENGTH_PER_CORE );

    asc_copy_ub2gm_sync(z + block_idx *
TILE_LENGTH_PER_CORE , zOut,
TILE_LENGTH_PER_CORE );
}
```

示例二：基于异步搬运/计算+统一同步接口

```
__global__ __aicore__ void add(__gm__ float* x,
__gm__ float* y, __gm__ float* z)
{
    KERNEL_TASK_TYPE(KERNEL_TYPE_AIV_ONLY);
    asc_init ();

   __ubuf__ float xIn[TILE_LENGTH_PER_CORE ];
   __ubuf__ float yIn[TILE_LENGTH_PER_CORE ];
   __ubuf__ float zOut[TILE_LENGTH_PER_CORE];

     asc_copy_gm2ub(xIn, x + block_idx *
TILE_LENGTH_PER_CORE, TILE_LENGTH_PER_CORE);
     asc_copy_gm2ub(yIn, y + block_idx *
TILE_LENGTH_PER_CORE, TILE_LENGTH_PER_CORE);
     asc_sync();

     asc_add(zOut, xIn, yIn, TILE_LENGTH_PER_CORE );
     asc_sync();

     asc_copy_ub2gm(z + block_idx *
TILE_LENGTH_PER_CORE , zOut,
TILE_LENGTH_PER_CORE );
     asc_sync();
}
```

示例三：基于异步搬运/计算+ 精细化同步接口

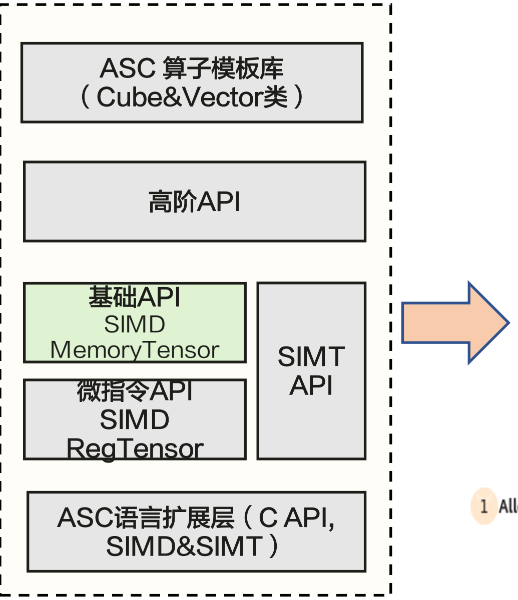
```
__global__ __aicore__ void add(__gm__ float* x, __gm__
float* y, __gm__ float* z)
{
    KERNEL_TASK_TYPE(KERNEL_TYPE_AIV_ONLY);
    asc_init ();

   __ubuf__ float xIn[TILE_LENGTH_PER_CORE ];
   __ubuf__ float yIn[TILE_LENGTH_PER_CORE ];
   __ubuf__ float zOut[TILE_LENGTH_PER_CORE ];

   asc_sync_notify(PIPE_V, PIPE_MTE2, EVENT_ID0);
   for (int i = 0; i < 2; i++) {
       asc_sync_wait(PIPE_V, PIPE_MTE2, EVENT_ID0);
       asc_copy_gm2ub(xIn, x + block_idx *
TILE_LENGTH_PER_CORE, TILE_LENGTH_PER_CORE);
       asc_copy_gm2ub(yIn, y + block_idx *
TILE_LENGTH_PER_CORE, TILE_LENGTH_PER_CORE);

       asc_sync_notify(PIPE_MTE2, PIPE_V, EVENT_ID0);
       asc_sync_wait(PIPE_MTE2, PIPE_V, EVENT_ID0);

       asc_add(zOut, xIn, yIn, TILE_LENGTH_PER_CORE );
       asc_sync_notify(PIPE_V, PIPE_MTE3, EVENT_ID0);
       asc_sync_wait(PIPE_V, PIPE_MTE3, EVENT_ID0);

       asc_sync_notify(PIPE_3, PIPE_MTE2, EVENT_ID0);
       asc_copy_ub2gm(z + block_idx *
TILE_LENGTH_PER_CORE , zOut, TILE_LENGTH_PER_CORE );
   }
   asc_sync_wait(PIPE_V, PIPE_MTE2, EVENT_ID0);
}
```

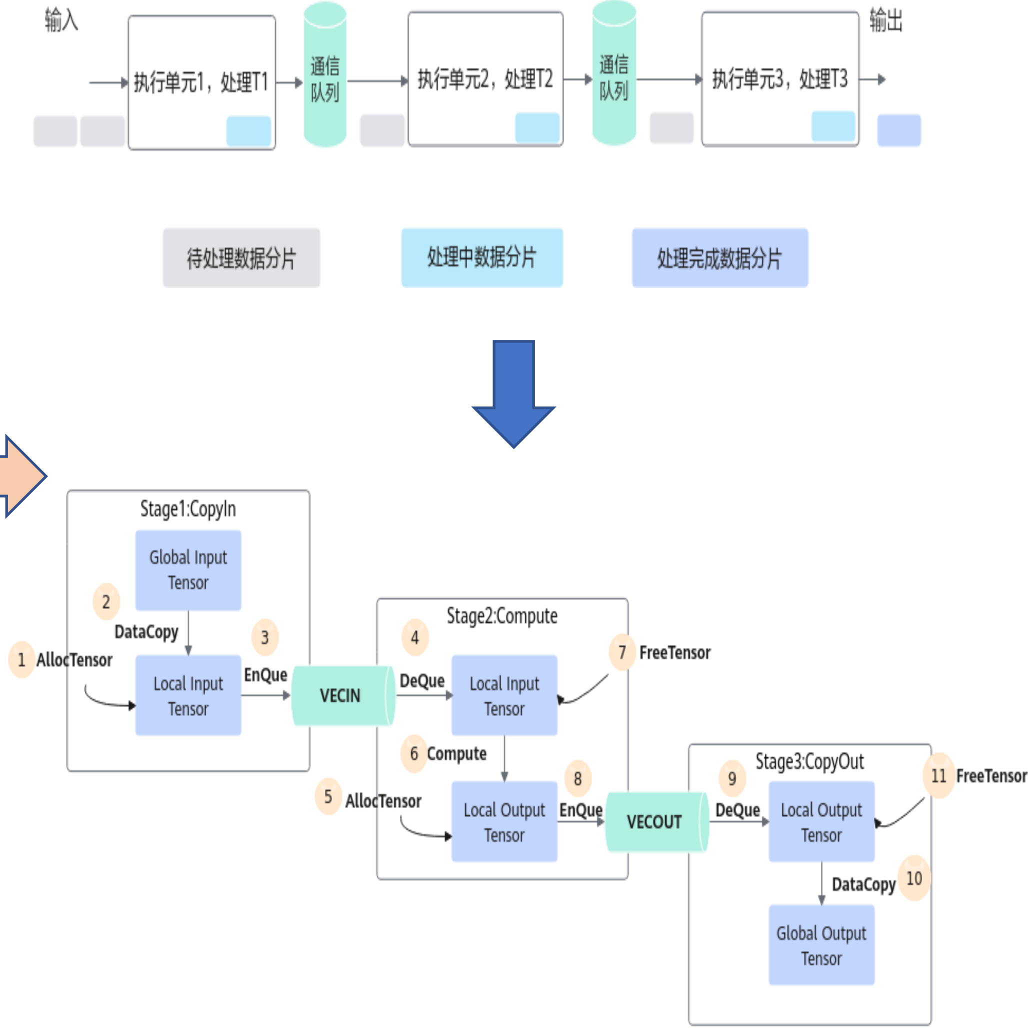
A2/A3 语言扩展层C API将逐步支持完善，年底支持完备向量编程

https://gitcode.com/cann/asc-devkit

Ascend C编程范式(2)：基于基础API的Tque/Tpipe范式Add算子示例

基于Tque/Tpipe抽象硬件同步和内存管理，简化编程

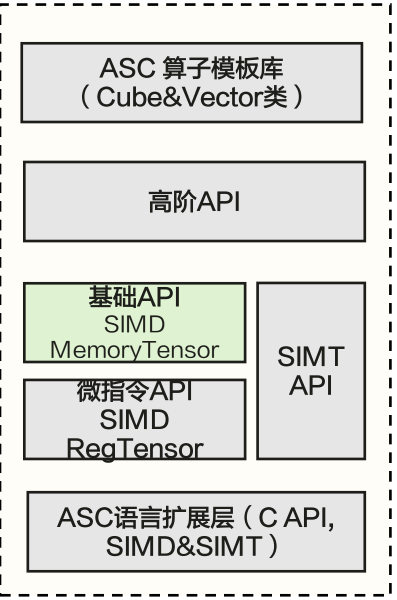





```cpp
    __aicore__ inline void Process()
    {
        int32_t loopCount = this->tileNum * BUFFER_NUM;
        for (int32_t i = 0; i < loopCount; i++) {
            CopyIn(i);
            Compute(i);
            CopyOut(i);
        }
    }

private:
    __aicore__ inline void CopyIn(int32_t progress)
    {
        AscendC::LocalTensor<float> xLocal = inQueueX.AllocTensor<float>();
        AscendC::LocalTensor<float> yLocal = inQueueY.AllocTensor<float>();
        AscendC::DataCopy(xLocal, xGm[progress * this->tileLength], this->tileLength);
        AscendC::DataCopy(yLocal, yGm[progress * this->tileLength], this->tileLength);
        inQueueX.EnQue(xLocal);
        inQueueY.EnQue(yLocal);
    }
    __aicore__ inline void Compute(int32_t progress)
    {
        AscendC::LocalTensor<float> xLocal = inQueueX.DeQue<float>();
        AscendC::LocalTensor<float> yLocal = inQueueY.DeQue<float>();
        AscendC::LocalTensor<float> zLocal = outQueueZ.AllocTensor<float>();
        AscendC::Add(zLocal,xLocal, yLocal, this->tileLength);
        outQueueZ.EnQue<float>(zLocal);
        inQueueX.FreeTensor(xLocal);
        inQueueY.FreeTensor(yLocal);
    }
    __aicore__ inline void CopyOut(int32_t progress)
    {
        AscendC::LocalTensor<float> zLocal = outQueueZ.DeQue<float>();
        AscendC::DataCopy(zGm[progress * this->tileLength], zLocal, this->tileLength);
        outQueueZ.FreeTensor(zLocal);
    }
```

Ascend C编程范式(3)：基于基础API的自主管理内存/同步的Add算子示例

基于LocalMemoryAllocator、Barrier 自主管理内存和同步，提供更底层控制能力



As-Is: TQue 范式编程

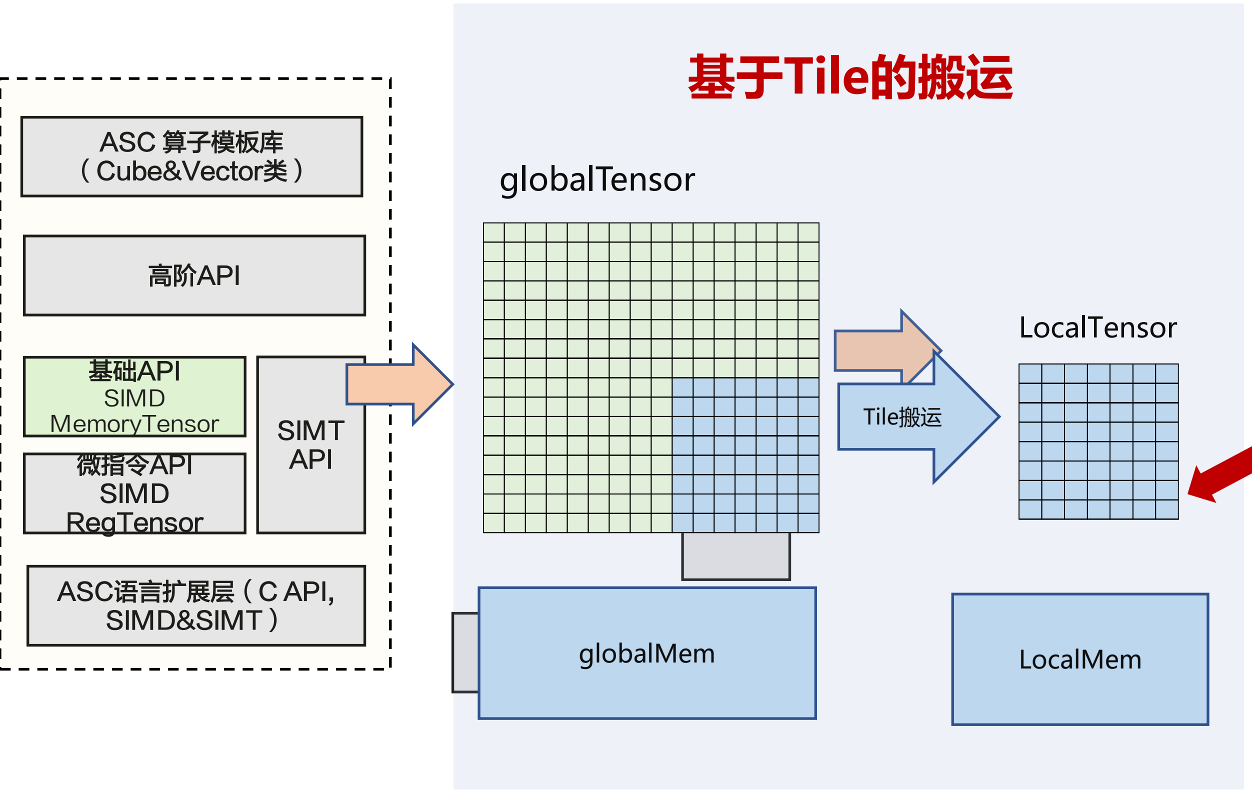
```
TPipe tpipe;
TQue<TPosition::VECIN, 1> vecInQue;
TQue<TPosition::VECOUT, 1> vecOutQue;

constexpr uint32_t tileLength = 1024;
// Initialize buffer size
tpipe.InitBuffer(vecInQue, 1,
tileLength*sizeof(half));
tpipe.InitBuffer(vecOutQue,1,
tileLength*sizeof(half));

for(int i=0; I < Loop; i++) {
  //  Allocate Tensor
 auto in = vecInQue.AllocTensor<half>();
 DataCopy(in, gm+ i*1024, 1024);

  // Insert sync
 vecInQue.EnQue(in);
 vecInQue.DeQue(in);

 auto out = vecOutQue.AllocTensor<half>();

 Abs(out, in, 1024);

 vecInQue.FreeTensor(in);

  // Insert sync
 vecOutQue.EnQue(out);
 vecOutQue.DeQue(out);
 DataCopy(gm+i*1024, out, 1024);
 vecOutQue.FreeTensor(out);
}
```

To Be: 用户底层控制，同步与内存分离

```
LocalMemAllocator<MemType::UB> allocator;

constexpr uint32_t tileLength = 1024;
auto inLocalTensor = allocator.Alloc<half>(tileLength);
auto outLocalTensor =
allocator.Alloc<half>(tileLength);


Barrier<SyncScope::INTRA_CORE, mte3, mte2>
firstBarrier;
Barrier<SyncScope::INTRA_CORE, mte2, pipe_v>
secBarrier;
Barrier<SyncScope::INTRA_CORE, pipe_v, mte3>
thirdBarrier;

firstBarrier.Arrive(0);

for(int i=0; i<loop; i++){
    firstBarrier.wait(0);
    DataCopy(in_ping, gm+ i *1024, 1024);
    secBarrier.ArriveAndWait(0);
    Abs(out_ping, in_ping, 1024);
    thirdBarrier.ArriveAndWait(0);
    DataCopy(gm+i*1024, out_ping, 1024);
    firstBarrier.Arrive(0);
}
firstBarrier.Wait(0);
```

Ascend C编程范式(4)：基于基础API Tensor Tile API编程

Tensor增加对Layout的支持，可提升编程的灵活性和可维护性，同时保持高性能；增强CUBE及VECTOR模板元编程能力；




```
// 1. 原始输入GM大小 M = 2048, N = 2048, K = 128
using GlobalTrait = TensorTrait<half, TPosition::GM, TwoDimLayout>;
auto aGlobal = GlobalTensor<GlobalTrait>(reinterpret_cast<__gm__ half *>(aGm) + blockOffset, LayoutDim2(Shape(M, K), Stride(K, 1)));
auto bGlobal = GlobalTensor<GlobalTrait>(reinterpret_cast<__gm__ half *>(bGm) + blockOffset, LayoutDIm2(Shape(K, N), Stride(N, 1)));
auto dstGlobal = GlobalTensor<GlobalTrait>(reinterpret_cast<__gm__ half *>(dstGm) + blockOffset, LayoutDIm2(Shape(M, N), Stride(N, 1)));
// 2. 定义静态片上内存
auto a1Local = AscendC::LocalTensor<TensorTrait<half, TPosition::A1, NZLayout<1024, 128>>>(0);
auto b1Local = AscendC::LocalTensor<TensorTrait<half, TPosition::B1, NZLayout<128, 1024>>>(128 * 1024 * sizeof(half));
auto a0Local = AscendC::LocalTensor<TensorTrait<half, TPosition::A0, SLayout<256, 128>>>(0);
auto b0Local = AscendC::LocalTensor<TensorTrait<half, TPosition::B0, SLayout<128, 256>>>(0);
auto c0Local = AscendC::LocalTensor<TensorTrait<half, TPosition::B0, SLayout<256, 256>>>(0);
// 3. 根据基本块的处理形态描述计算行为(只描述计算行为, 同步处理省略)
for (size_t mIndex = 0; mIndex < M; mIndex += 1024) {
    for (size_t nIndex = 0; nIndex < N; nIndex += 1024) {
        DataCopy(a1Local, aGlobal, Coord2(mIndex, 0));
        DataCopy(b1Local, bGlobal, Coord2(0, nIndex));
        for (size_t mInnerIndex = 0; mInnerIndex < 1024; mInnerIndex += 256) {
            for (size_t nInnerIndex = 0; nInnerIndex < 1024; nInnerIndex += 256) {
                for (size_t kInnerIndex = 0; kInnerIndex < 128; kInnerIndex += 128) {
                    LoadData(a0Local, a1Local, Coord2(mInnerIndex, kInnerIndex));
                    LoadData(b0Local, b1Local, Coord2(kInnerIndex, nInnerIndex));
                    Mmad(c0Local, a0Local, b0Local, {256, 128, 256});
                }
                FixPipe(dstGlobal[mIndex * 1024 * 2048 + nIndex * 1024], c0Local, CoordRowMajor(mInnerIndex, nInnerIndex));
            }
        }
    }
}
```

定义带layout的LocalTensor

基于tile的搬运和计算

ASC支持异构编译+<<<>>>直调，提供与业界一致编程体验

- 对标业界，支持异构融合编译和<<<>>>直调；
- 通过后缀名“.asc”或 编译选项“-x asc”使能异构编译；

```
#include "kernel_operator.h"
#include "acl/acl.h"
__global__ __aicore__ void
hello_world()
{
  AscendC::printf("Hello World!!!\n");
}
int32_t main(int argc, char const
*argv[])
{
  aclInit(nullptr);
  int32_t deviceId = 0;
  aclrtSetDevice(deviceId);
  aclrtStream stream = nullptr;
  aclrtCreateStream(&stream);
  constexpr uint32_t blockDim = 8;
  hello_world<<<blockDim, nullptr,
stream>>>();
  aclrtSynchronizeStream(stream);
  aclrtDestroyStream(stream);
  aclrtResetDevice(deviceId);
  aclFinalize();
  return 0;
}
```

Device侧代码

Host/Device编译归一

Host侧代码

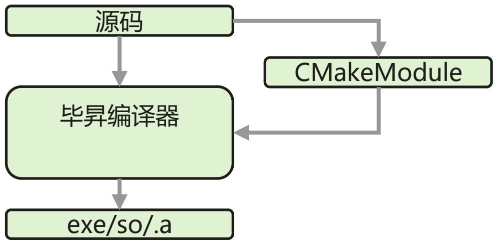
https://gitcode.com/cann/asc-devkit
https://gitcode.com/cann/asc-tools

使用命令行一键编译：

```
bisheng main.asc --npu-soc
Ascend910B1
```

或基于标准cmake脚本

```
project(hello LANGUAGES ASC)
find_package(ASC)

add_executable(hello
    hello_world.asc
)
```



CANN

ASC CPU孪生调试能力增强

- 一套代码同时支持CPU和NPU调试，通过编译选项使能；
- CPU 孪生调试可解决大多数代码逻辑错误，包括同步指令不匹配校验等；

As-Is

单独代码使能CPU孪生调试

简化CPU孪生调试使能

```
int32_t main(int32_t argc, char *argv[])
{
    uint32_t blockDim = 8;
    size_t inputByteSize = 8 * 2048 * sizeof(uint16_t);
    size_t outputByteSize = 8 * 2048 * sizeof(uint16_t);
#ifdef ASCENDC_CPU_DEBUG
    uint8_t *x = (uint8_t *)AscendC::GmAlloc(inputByteSize);
    uint8_t *y = (uint8_t *)AscendC::GmAlloc(inputByteSize);
    uint8_t *z = (uint8_t *)AscendC::GmAlloc(outputByteSize);

    ReadFile("./input_x.bin", inputByteSize, x, inputByteSize);
    ReadFile("./input_y.bin", inputByteSize, y, inputByteSize);

    AscendC::SetKernelMode(KernelMode::AIV_MODE);
    ICPU_RUN_KF(Add, blockDim, x, y, z); // use this macro for cpu debug

    WriteFile("./output_z.bin", z, outputByteSize);

    AscendC::GmFree((void *)x);
    AscendC::GmFree((void *)y);
    AscendC::GmFree((void *)z);
#else
    CHECK_ACL(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_ACL(aclrtSetDevice(deviceId));
    aclrtStream stream = nullptr;
    CHECK_ACL(aclrtCreateStream(&stream));

    uint8_t *xHost, *yHost, *zHost;
```

孪生调试用户编写的特有代码

To Be

一套代码，通过编译选项区分CPU孪生调试和NPU上板调试

```
int32_t main(int32_t argc, char *argv[])
{
    uint32_t blockDim = 8;
    size_t inputByteSize = 8 * 2048 * sizeof(uint16_t);
    size_t outputByteSize = 8 * 2048 * sizeof(uint16_t);
    CHECK_ACL(aclInit(nullptr));
    int32_t deviceId = 0;
    CHECK_ACL(aclrtSetDevice(deviceId));
    aclrtStream stream = nullptr;
    CHECK_ACL(aclrtCreateStream(&stream));

    uint8_t *xHost, *yHost, *zHost;
    uint8_t *xDevice, *yDevice, *zDevice;

    CHECK_ACL(aclrtMallocHost((void **)(&xHost), inputByteSize));
    CHECK_ACL(aclrtMallocHost((void **)(&yHost), inputByteSize));
    CHECK_ACL(aclrtMallocHost((void **)(&zHost), outputByteSize));
    CHECK_ACL(aclrtMalloc((void **)&xDevice, inputByteSize, ACL_MEM_MALLOC_HUGE_FIRST));
    CHECK_ACL(aclrtMalloc((void **)&yDevice, inputByteSize, ACL_MEM_MALLOC_HUGE_FIRST));
    CHECK_ACL(aclrtMalloc((void **)&zDevice, outputByteSize, ACL_MEM_MALLOC_HUGE_FIRST));
    ReadFile("./input_x.bin", inputByteSize, xHost, inputByteSize);
    ReadFile("./input_y.bin", inputByteSize, yHost, inputByteSize);

    CHECK_ACL(aclrtMemcpy(xDevice, inputByteSize, xHost, inputByteSize, ACL_MEMCPY_HOST_TO_DEVICE));
    CHECK_ACL(aclrtMemcpy(yDevice, inputByteSize, yHost, inputByteSize, ACL_MEMCPY_HOST_TO_DEVICE));

    Add<<<blockDim, nullptr, stream>>>(xDevice, yDevice, zDevice);
    CHECK_ACL(aclrtSynchronizeStream(stream));

    CHECK_ACL(aclrtMemcpy(zHost, outputByteSize, zDevice, outputByteSize, ACL_MEMCPY_DEVICE_TO_HOST));
    WriteFile("./output_z.bin", zHost, outputByteSize);
```

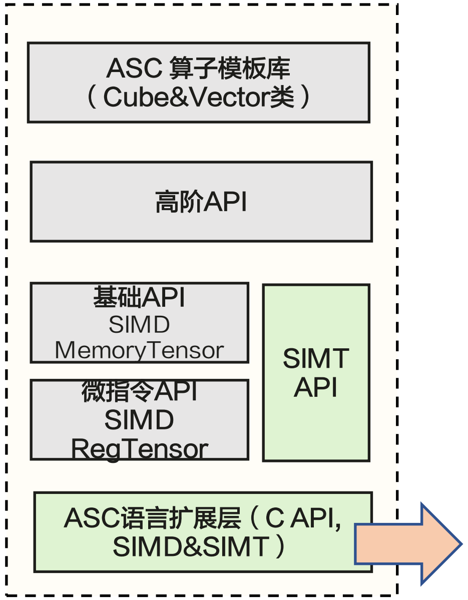
执行代码一致，用户不用写而外代码

https://gitcode.com/cann/asc-tools/

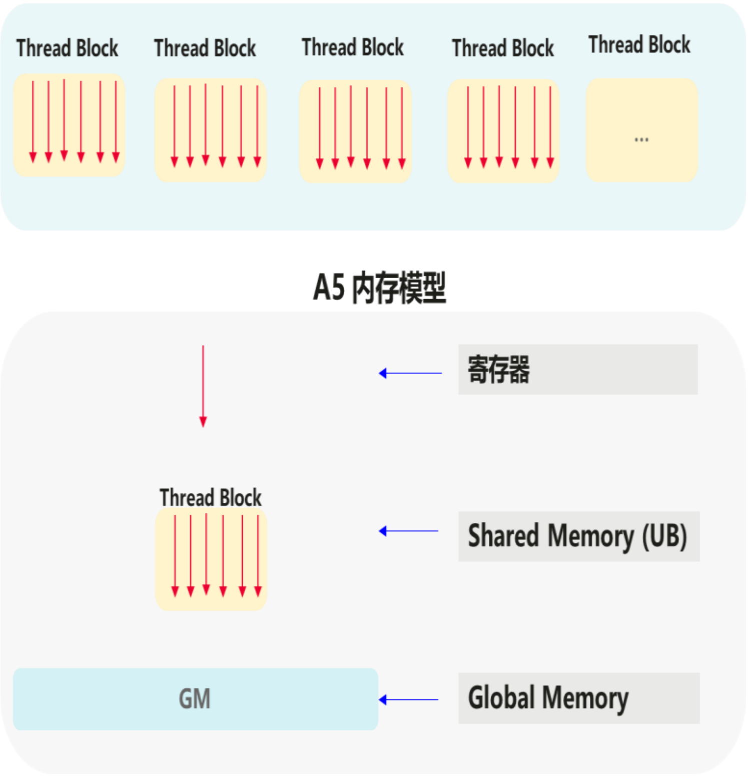
目录

ASC 基于新一代硬件支持SIMT编程

支持与业界类似的SIMT编程体验，优化离散和复杂逻辑处理等场景算子开发



SIMT 编程模型



SIMT 算子示例

```
template <typename T>
__global__LAUNCH_BOUND(512) inline void VectorAdd(T* dst, T*
src0, T* src1, uint32_t len)
{
   // Just show how to allocate share memory
   __ubuf__ T sharedBuf[512];

   //  blockDim3 will be supported soon
   int idx = blockDim.x * blockIdx.x + threadIdx.x;

  // Just show how to use share memory
  sharedBuf[threadIdx.x] = src0[idx] + src1[idx];

   if (idx < len) {
      dst[idx] = sharedBuf[threadIdx.x];
   }
   ...
   asc_syncthreads();
   ...
}
```

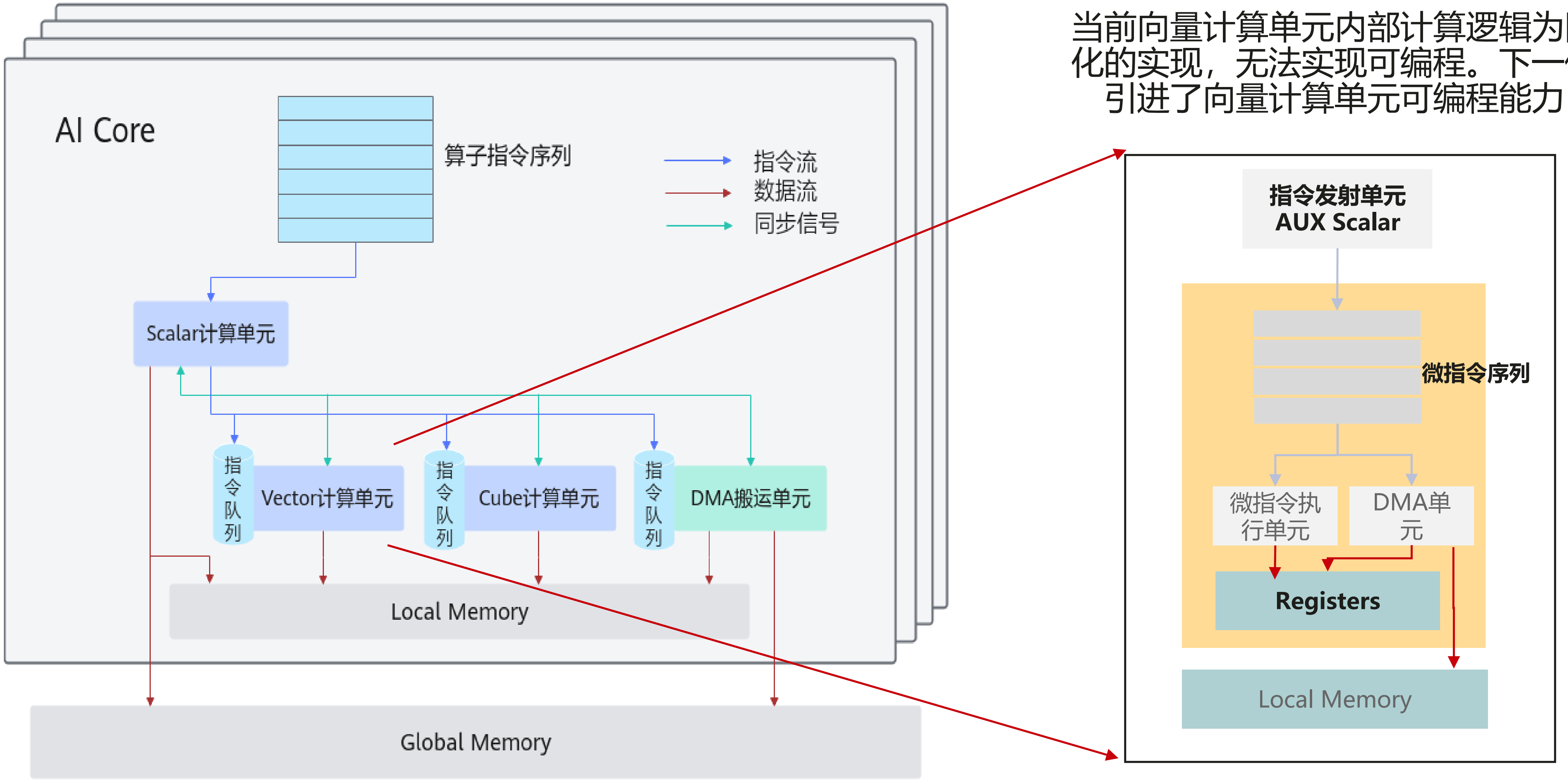
开放SIMT编程，适合处理离散访问、复杂控制逻辑处理等场景的优势

CANN

下一代SIMD 抽象架构的演进

持续优化SIMD架构，基于寄存器架构更好发挥SIMD优势

当前向量计算单元内部计算逻辑为固化的实现，无法实现可编程。下一代引进了向量计算单元可编程能力

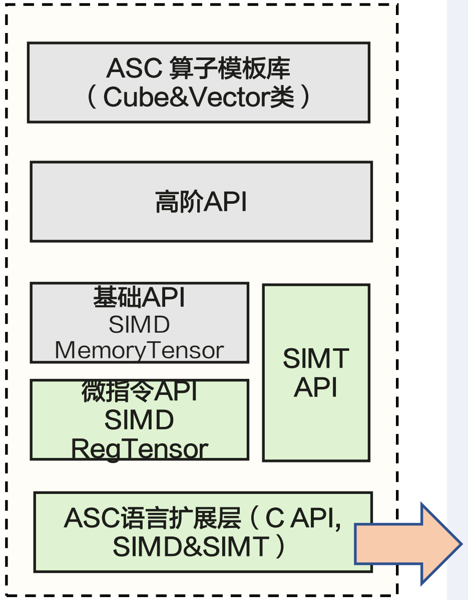


Vector可编程计算单元

CANN

ASC 基于新一代支持SIMT& SIMD新同构编程

同一个算子Kernel同时支持SIMD&SIMT，充分发挥各自优势



新同构SIMT示例

```
template <typename T>
__simt_vf__ __aicore__
LAUNCH_BOUND(512) inline void
SimtReduce(
__ubuf__ T* dst, __ubuf__ T* src0, uint32_t
len)
{
    __ubuf__ T shareData;
    int  idx = blockDim.x * blockIdx.x +
threadIdx.x;

    if (idx < len) {
        shareData[threadIdx.x] = src0[idx];
    }
    asc_syncthreads();

    ...

    if (threadIdx.x == 0) {
        dst[blockIdx.x] = shareData[0];
    }
}
```

开放SIMT编程，适合处理离散访问、复杂控制逻辑处理等场景的优势

新同构SIMD示例

```
__simd_vf__ __aicore__ inline void
SimdAdd(__ubuf__ half* dstPtr, __ubuf__
  half * src0Ptr, __ubuf__ half * src1Ptr, uint32_t len
) {
    constexpr uint16_t oneRepeatSize =
asc_get_vf_len() / sizeof(half);
    uint16_t repeatTimes = ceil(len, oneRepeatSize);

    vector_half dstReg0, srcReg0, srcReg1;
    vector_bool maskReg;

    for (uint16_t i = 0; i < repeatTimes; i++) {
        maskReg = asc_update_mask(len);
        asc_load(srcReg0, src0Ptr + i *
oneRepeatSize);
        asc_load(srcReg1, src1Ptr + i *
oneRepeatSize);
        asc_add(dstReg0, srcReg0, srcReg1, maskReg);
        asc_store(dstPtr + i * oneRepeatSize, dstReg0,
maskReg);
    }
}
```

开放SIMD微指令API，直接操作VEC向量寄存器编程，适合连续规整计算

SIMT&SIMD混合调用示例

通过VF_CALL调用SIMT和SIMT函数

```
template<typename T>
extern "C" __global__ __aicore__ inline void
MixKernel(__gm__ T* x, __gm__ T *  y, __gm__
T * z, uint32_t len)
{

  ...
  // Execute SIMT Vector Function(VF) B
   asc_call_vf<SimtReduce<T>>(blockDim,
dst0Local, src0Local, src1Local, len);

   // Execute SIMD Vector Function(VF)
   asc_call_vf <SimdAdd>(dst0Local,
src0Local, src1Local, len);
}
```

基于SIMT&SIMD新同构编程，充分发挥各自的优势

Ascend C 规划路线图

Ascend C 全面开源

11/28

- 支持A2/A3 语言扩展层C API初版
- 支持Tensor Tile API初稿
- 基础API全面开源

2025/12/30

- 支持A2/A3 语言扩展层C **API，支撑向量开发**
- 支持Tensor Tile API
- 完善样例等

2026上半年

- 全量支持A2/A3 C API
- 基于下一代支持**SIMT编程**
- 基于下一代支持**SIMD&SIMT**新同构编程
- 基础API 完善Tensor Tile API
- 提升调试调优效率
- Python前端支持SIMT、Tensor 编程能力

2026下半年

- 完善设备侧类库的支持
- 完善Python前端编程能力

...

- 适配新硬件，提供更优编程体验

https://gitcode.com/cann/asc-devkit
https://gitcode.com/cann/asc-tools

欢迎交流 & 贡献

asc-devkit 代码仓

asc-tools 代码仓

https://gitcode.com/cann/asc-devkit
https://gitcode.com/cann/asc-tools

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态
社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem
Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.

上CANN社区获取干货

关注CANN公众号获取资讯

https://gitcode.com/cann

CANN