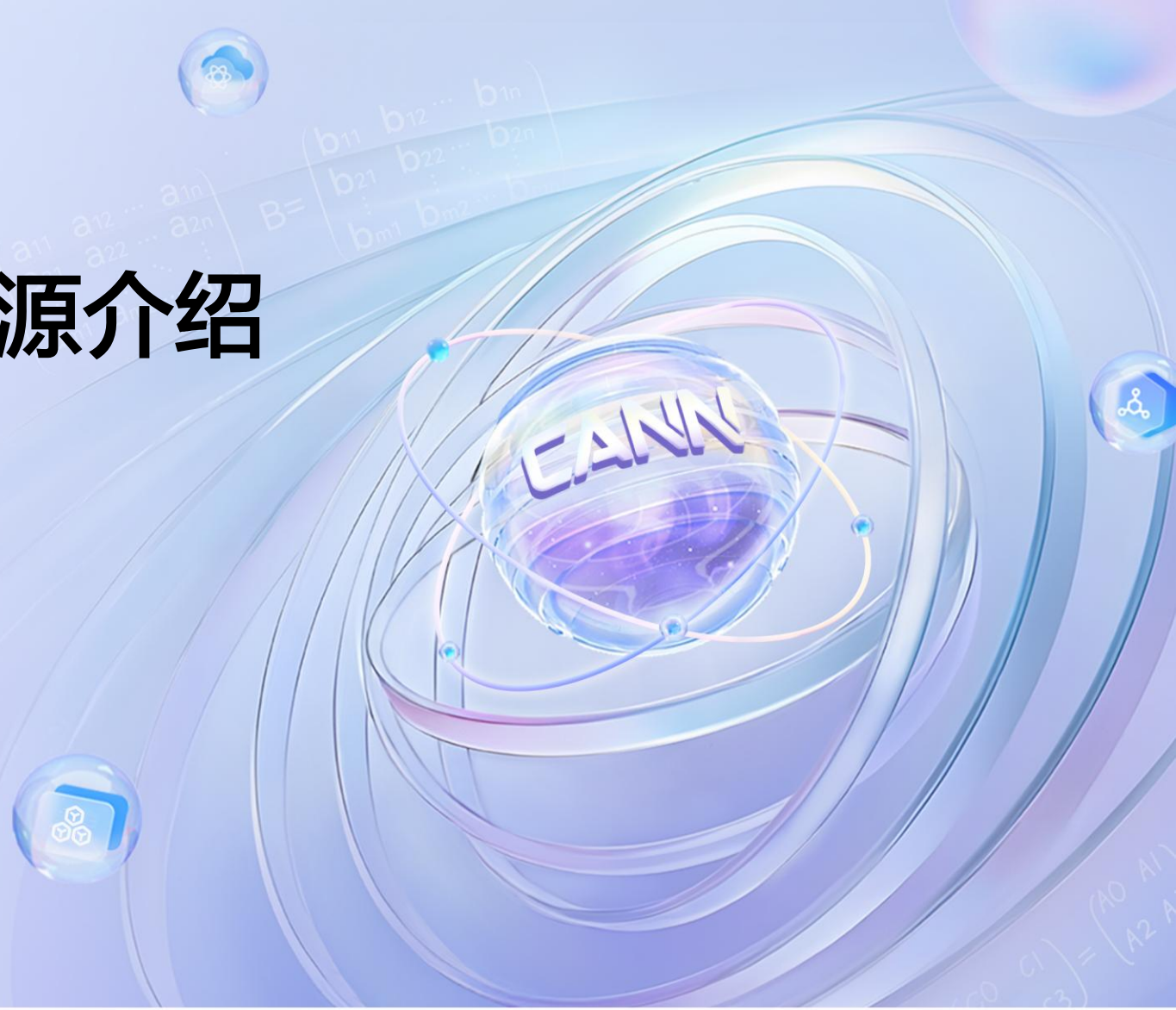
CANN Runtime开源介绍

作者：张子菁

时间：2025/12/25



CANN

什么是CANN运行时

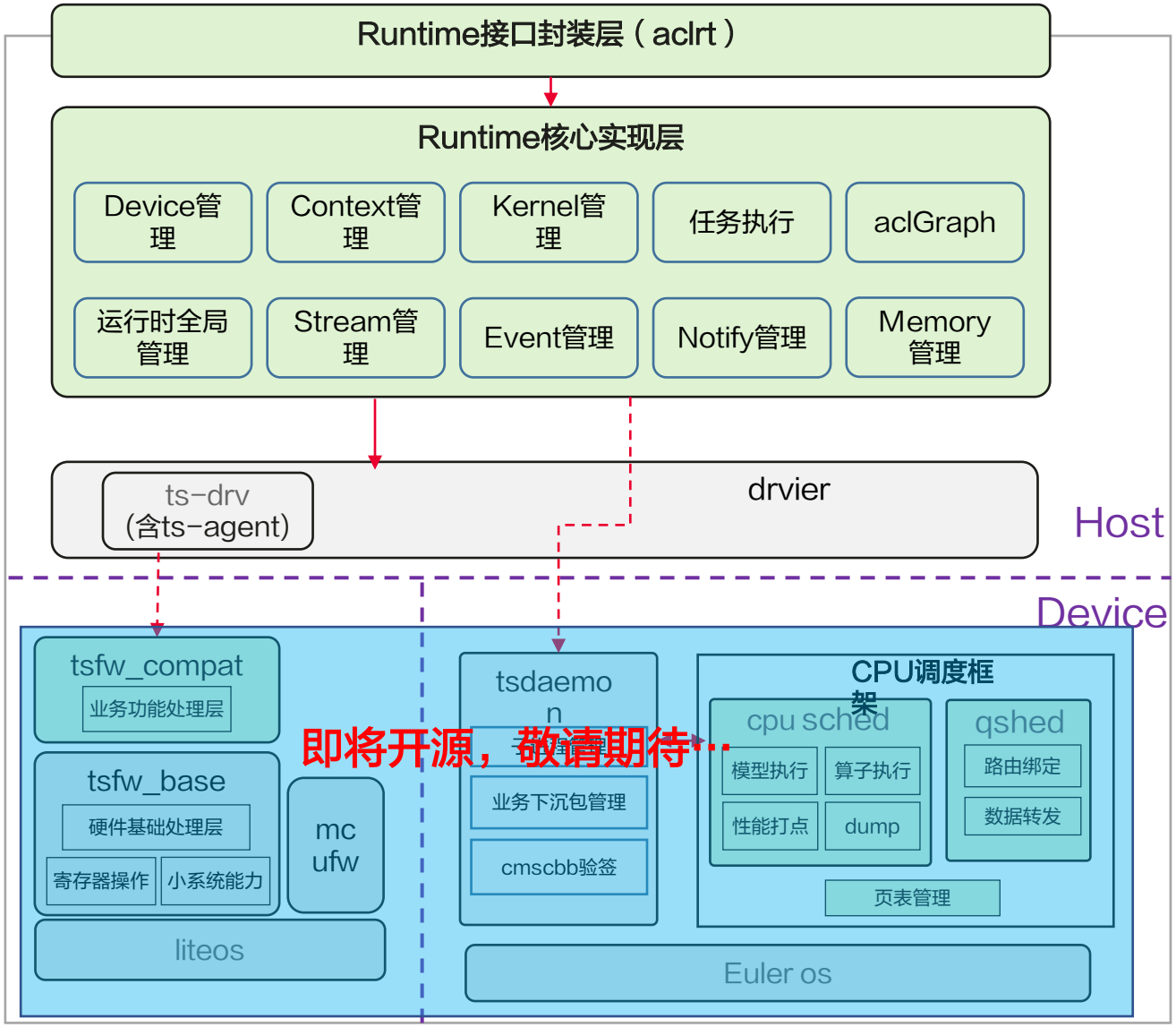
CANN运行时是华为昇腾平台 CANN 软件栈中负责驱动硬件执行与管理 AI 计算任务的核心组件，它通过提供统一的运行环境与 API，使得上层应用和框架能够高效利用 Ascend NPU 计算资源。其**主要功能**包括以下几类：

- **设备与上下文管理**：负责识别和初始化 NPU 设备，创建计算上下文（Context），并在多进程/多线程场景中管理资源状态与隔离。
- **流（Stream）与事件（Event）调度**：支持异步计算流与事件同步机制，允许多个计算任务并行执行并根据依赖关系进行精细调度。
- **内存管理与数据传输**：管理主机与 NPU 之间，以及 NPU 内部的内存分配和数据拷贝，优化片上内存使用以提高带宽利用效率。
- **核函数（Kernel）管理**：核函数管理负责核函数的加载、缓存、调度与执行控制，统一算子执行入口，保障多流并发下的高效运行与稳定性。

简单来说，CANN 运行时是**连接应用与 NPU 硬件的桥梁**，实现从资源初始化、任务调度、内存通信到模型执行的完整运行时支持。

CANN

CANN运行时核心功能



Device管理：NPU设备，提供对设备set/reset，设备查询，设备配置等功能
Context管理： 是NPU设备的执行上下文环境。提供上下文创建、销毁，切换和配置等功能
Stream管理：提供下发给NPU的任务按序执行功能。提供流创建、销毁；流属性查询、配置以及流同步，流状态管理功能
Event管理：用于设备内流间同步的事件。提供Event创建、销毁、事件同步等功能。
Notify管理：作为Event资源的补充，提供跨卡同步功能。提供Notify创建、销毁、事件同步等功能
Memory管理：提供设备内存，Host内存管理的申请、释放，内存的拷贝功能。
Kernel管理：aic/aiv/aicpu算子注册管理和KernelLaunch功能
aclGraph：提供capture或build方式构建运行时aclGraph图，利用图中任务批量下发能力，节省host调度耗时提升整体性能
运行时全局管理：提供运行时初始化和进程级配置，DFX配置和全局管理功能

TSFW-compat：是tsfw的可升级插件，提供任务更新，资源管理和维测功能。
TSFW-base：负责硬件调度器启动初始化和基础处理流程

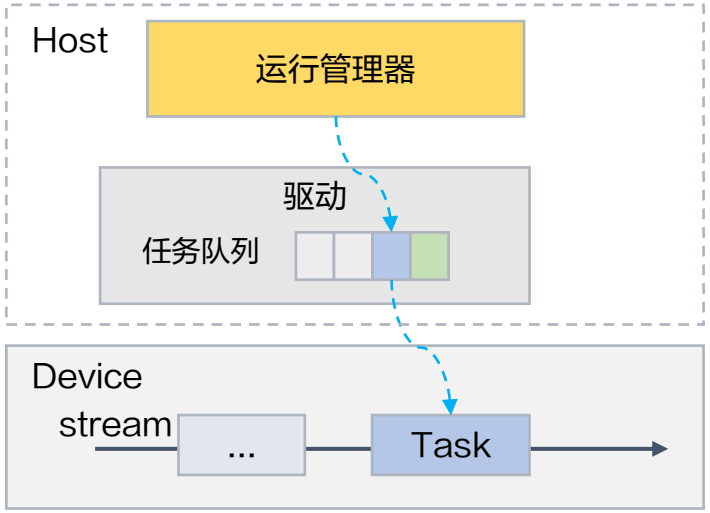
CPU调度器：提供数据面事件（包括CPU算子任务事件）分发处理，CPU算子执行等功能。
数据队列调度器：提供队列数据转发调度功能，用于data-driven模型执行的场景。

CANN

CANN运行时特点--极致性能的软硬协同调度

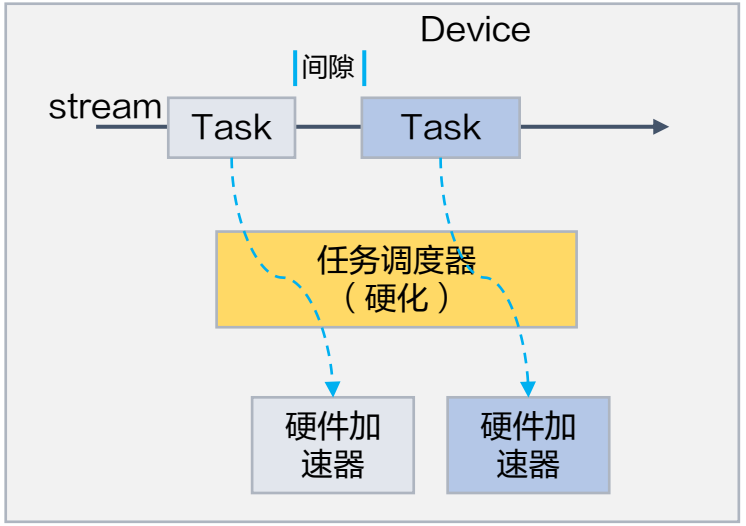
高性能任务下发

极致性能任务下发（~3us）



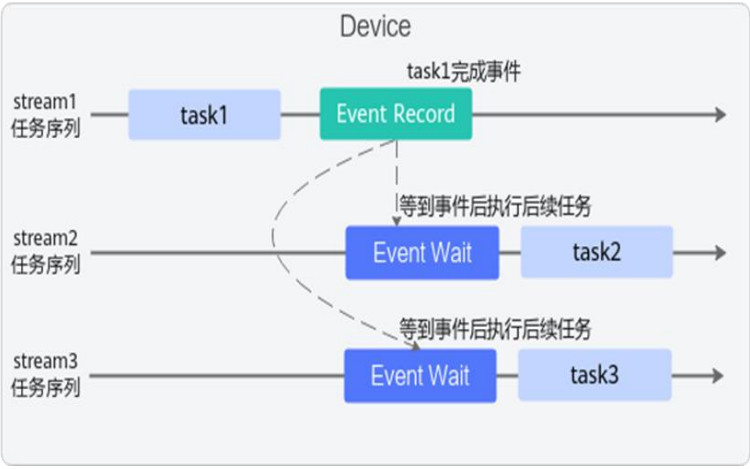
高效任务调度

任务间调度间隙<0.1us



高效同步机制

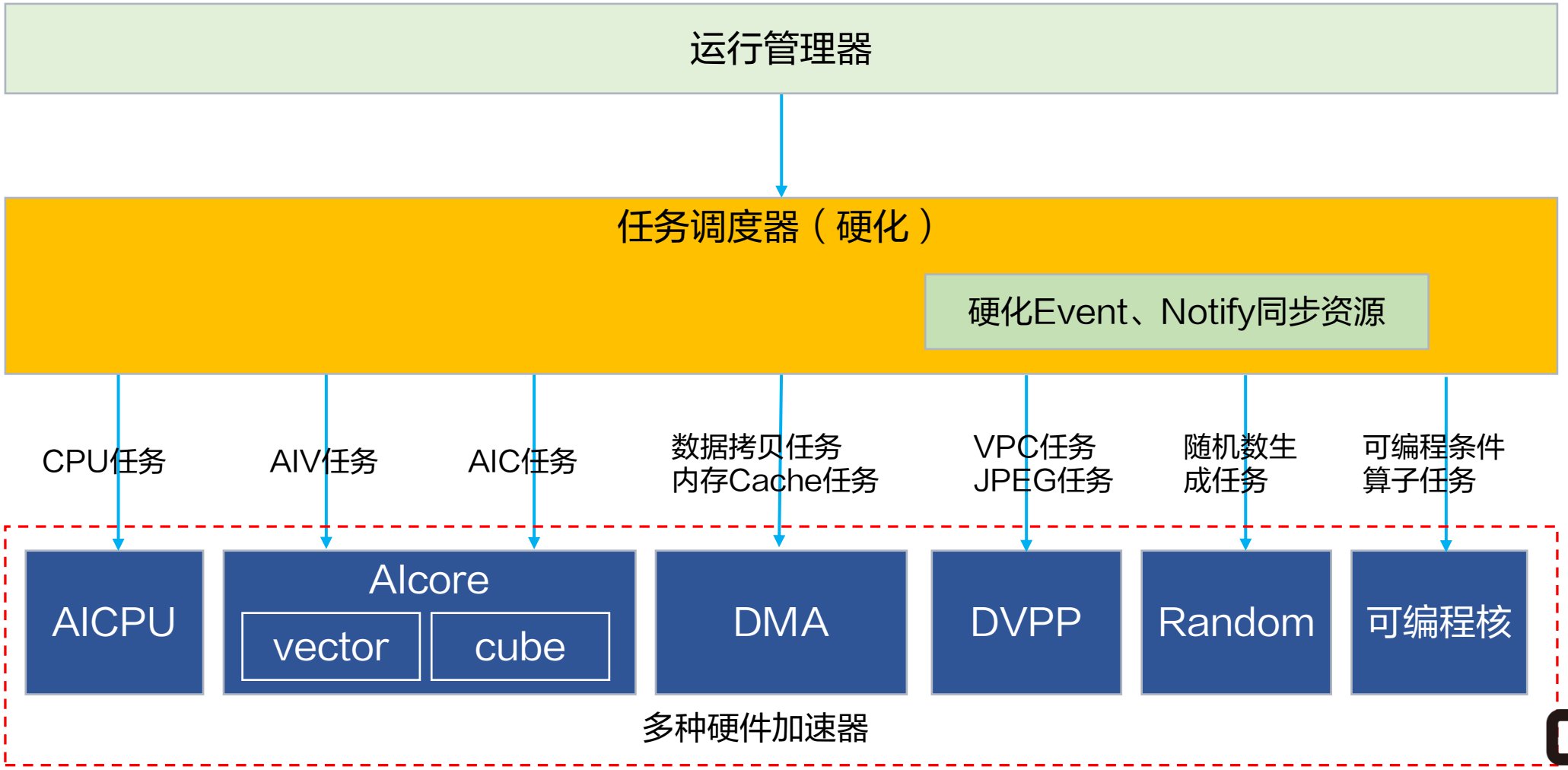
提供硬化Event同步资源，流间任务同步极致时延<100ns（少量情况600ns)



CANN

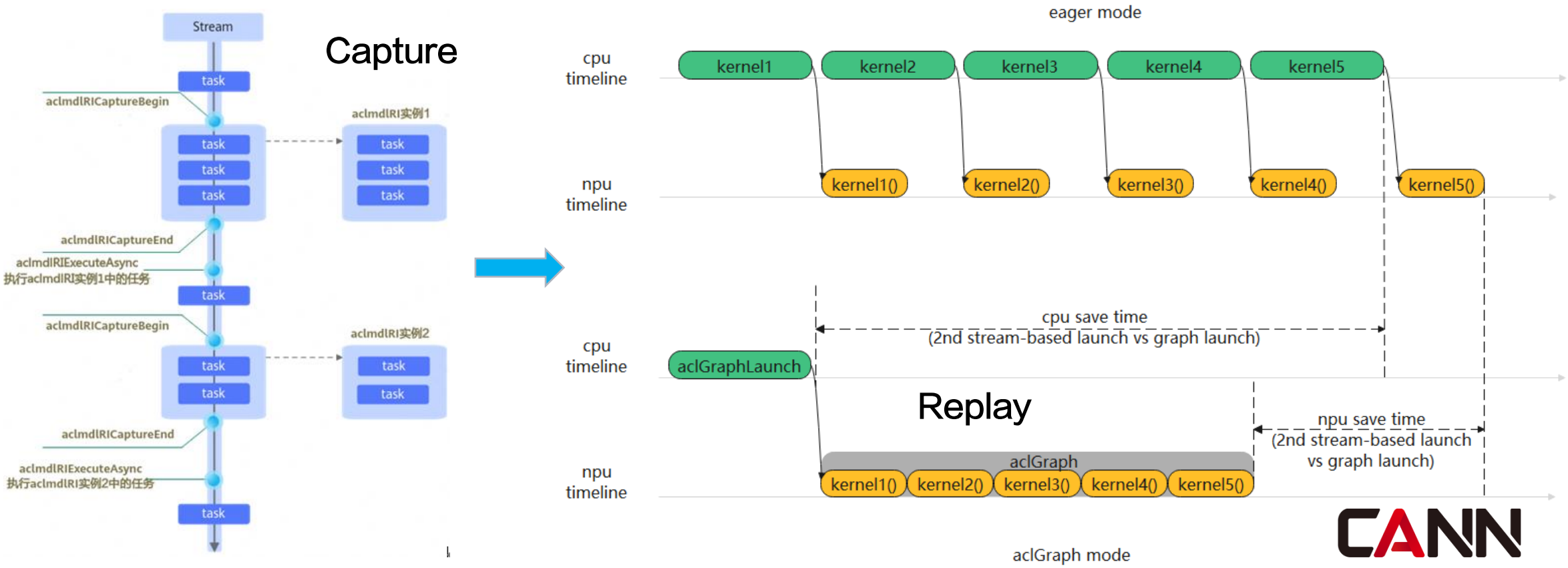
CANN运行时特点--协同调度多种硬件加速器

- 支持在单条流（Stream)上，混合排布不同硬件加速器的任务，任务调度器统一保序调度
- 资源感知调度：识别各硬件加速器忙闲状态，支持使用不同硬件加速器多条流之间并发执行。



CANN运行时特点--模型下沉执行

- **支持Capture模式：**在原有的stream下发任务代码流程中，增加aclmdlRICaptureBegin和aclmdlRICaptureEnd把stream上的task捕捉成下沉模型。
- **低延迟执行：**模型预下沉device，减小host任务下发时延，支持低延迟场景，提升实时响应能力。
- **解决HostBound：**将调度过程区分为下发capture+执行replay两阶段，通过采用1次capture+多次replay的执行模式减小H2D交互开销



为什么开源CANN运行时

•加速技术迭代与创新：
开源后，CANN Runtime 将不断收到来自全球开发者的反馈与建议，快速适应行业需求，促进技术的演化与创新。

•共享AI计算的底层能力：
通过社区的力量，CANN Runtime 将为更多AI应用提供稳定、高效的底层计算支持，推动 AI 产业技术的普及与落地。

透明化：彻底了解底层实现

•完全开放的代码：CANN Runtime 开源后，开发者能够直接查看其底层实现，包括任务调度、内存管理、硬件加速技术等关键机制。这样不仅可以深入理解 Runtime 的内部结构和工作原理，还能够快速识别和定位性能瓶颈及优化点。

•探索计算的细节：每一行代码的透明展示，帮助开发者理解如何高效地管理 AI 任务的执行，优化计算资源的使用，并且让调试与性能调优变得更加直接和可控。

可定制性：打造因地制宜的运行时环境

•灵活定制功能：开源使得开发者能够根据自己的特定需求定制和调整 CANN Runtime。例如，针对不同的硬件平台，开发者可以根据硬件特性调优内存管理、任务调度和执行流的策略。

•跨平台支持：在支持多硬件平台的基础上，开发者可以根据需要为新平台（如新处理器、加速器等）设计专有的运行时优化，满足多种部署需求。

•精准优化：对于不同规模的 AI 工作负载，开发者可以自定义执行策略、优化内存使用，并在高度并行的场景中精细调整性能。

生态共建：携手推动 AI 计算未来

•社区合作与贡献：CANN Runtime 的开源为开发者和技术爱好者提供了一个开放的合作平台。任何人都可以提交问题、解决 bug 或贡献新功能。通过开放的开发流程，开发者可以在社区中互相学习、共享经验，并共同推动技术的演进。

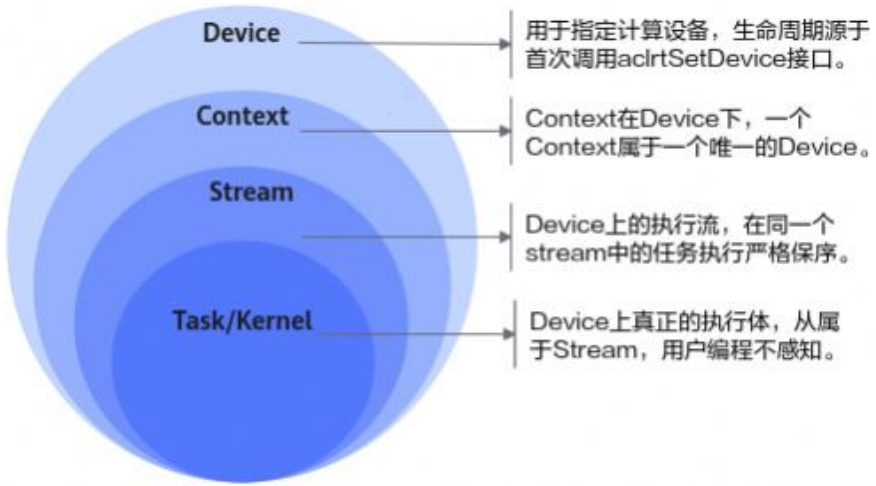
•技术创新与前沿探索：随着开源社区的逐渐壮大，CANN Runtime 将成为 AI 计算领域创新的试验场。开发者可以在这里尝试新技术、提出新需求，甚至影响未来的技术方向。

•全球技术力量聚集：通过开源，CANN Runtime 吸引全球开发者参与，这将加速技术的成熟与完善，并让更多有潜力的技术和解决方案得到广泛应用。

CANN运行时提供的编程模型

编程模型假定系统由 **主机（Host）**和 **设备（Device）**组成，二者拥有彼此独立的内存空间。
Host：指与Device相连接的服务器，会利用Device提供的NN（Neural-Network）计算能力，完成业务。
Device（或称NPU）：指安装了昇**腾AI处理器**的硬件设备，利用总线（PCIe，HCCS)与Host侧连接，提供NN计算能力。

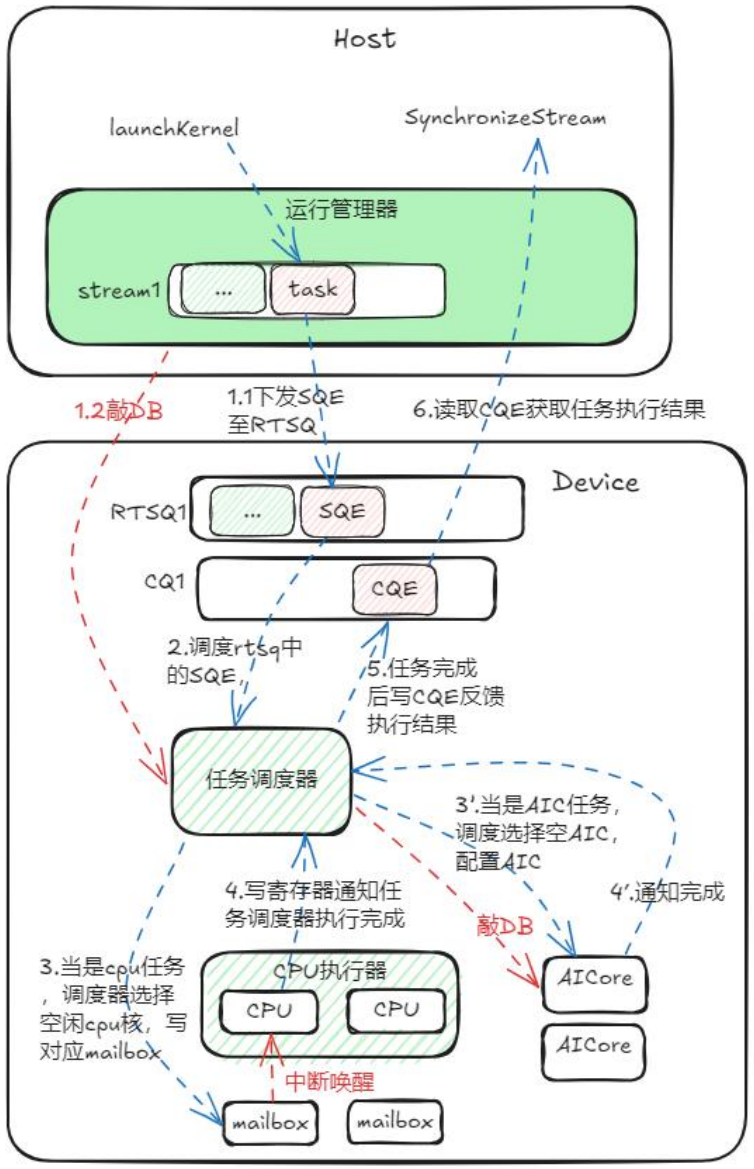
Device资源和调度按如下图所示模型进行管理



- **Device**，表示NPU计算设备。运行时提供acl接口管理设备生命周期（包含设备的打开、重置以及异常状态处理）
- **Context**，在Device下的运行上下文，context提供对stream等运行资源的隔离管理。一个Context唯一属于Device；可以显式地为一个Device创建多个context。
  - Context分隐式创建和显式创建
  - 隐式创建的Context（即默认Context），调用aclrtSetDevice接口会隐式创建默认Context。
  - 显式创建的Context，调用aclrtCreateContext接口会显式创建Context，调用aclrtDestroyContext接口显式销毁Context。
  - 进程内的Context可被所有线程可见，某个线程关联的context可以通过aclrtGetCurrentContext、aclrtSetCurrentContext接口进行切换。
- **Stream，是Device上的执行任务序列抽象，在同一个stream中的任务执行严格保序。**
  - Stream分隐式创建和显式创建。
  - 每个Context会包含一个默认Stream，这个属于隐式创建。
  - 用户可以显式创建Stream，调用aclrtCreateStream接口显式创建Stream，调用aclrtDestroyStream接口显式销毁Stream。显式创建的Stream归属当前线程关联的Context。
- **Task/Kernel，是Device上任务执行体，**NPU任务主要分为计算类任务，内存拷贝，通信类任务等。

CANN

典型执行流程



接口调用	处理
aclrtSetDevice(devId)	初始化Device对象： 1. 创建并初始化Device对象 2. 为Device创建默认context; 3. 为默认context创建默认流; 4. 启动device侧CPU执行器(进程)。
aclrtCreateStream(&stream1)	在当前context下创建流： 1. 通过驱动和TSFW配合为流分配任务调度器的RTSQ; 2. 创建stream对象资源并与RTSQ关联。 3. stream纳入到当前context管理。
aclrtMalloc(&devPtr, size)	申请设备内存 1. 申请host进程空间VA地址 2. 申请device侧物理内存 3. 建立VA与物理内存的页表（CPU执行器，AIC等硬件SMMU共页表）
aclrtMemcpy(devPtr, size, hostPtr, size, ACL_MEMCPY_HOST_TO_DEVICE)	H2D内存同步拷贝 1. 根据源、目的地址构造DMA描述符后下发DMA任务 2. 等待DMA任务完成，接口返回
myKernel<<<blockdim, stream1>>>(devPtr)	stream下发myKernel任务 1. myKernel的二进制代码注册并加载至device (仅首次） 2. 将myKernel的执行参数拷贝至device 3. 分配任务管理资源，并创建任务描述符 4. 下发任务到此流对应的RTSQ 5. 下发完任务即接口异步返回
	任务调度器开始调度 1. 调度器按RTSQ的优先级采用SP方式调度 2. 解析RTSQ中下发的任务描述符 3. 根据任务类型判断对应加速单元是否有空闲 4. 有空闲且满足任务的核数量要求，则将任务调度给对应加速单元（CPU算子会调度给CPU执行器） 5. 加速单元执行任务 6. 调度器等待加速单元执行完成，刷新任务状态和加速单元忙闲状态。当任务执行异常，硬件调度器会上报中断给TSFW，由TSFW异常处理。 7. 继续调度其他任务
aclrtSynchronizeStream(stream1)	流同步 1. 接口会阻塞当前CPU线程 2. 通过轮循+中断机制同步流执行状态 3. 流上任务全部完成，此接口返回。
aclrtMemcpy(hostPtr, size, devPtr, size, ACL_MEMCPY_DEVICE_TO_HOST)	D2H将结果拷贝回host

CANN

设备管理

运行时提供NPU生命周期管理和设备查询功能。可以支持使用单设备，多设备不同场景。

1. 单Device使用

```
// 指定Device 0作为计算设备
aclError ret = aclrtSetDevice(0);
// 执行任务1
......
// 复位Device 0，释放计算资源
ret = aclrtResetDeviceForce(0);
```

2. 多Device使用

```
// 指定Device 0作为计算设备
aclError ret = aclrtSetDevice(0);
// 执行任务1
......
// 指定Device 1作为计算设备
ret = aclrtSetDevice(1);
// 执行任务2
......
// 复位Device 0，释放计算资源
ret = aclrtResetDeviceForce(0);
// 复位Device 1，释放计算资源
ret = aclrtResetDeviceForce(1);
```

Host附带Device数量不同，Device硬件配置也不同，可以通过以下接口查询每个Device信息。

- 调用aclrtGetSocName接口获取当前运行环境的昇腾AI处理器型号。
- 调用aclrtGetDeviceCount接口获取Device数量，可用Device ID的取值范围：[0, (Device数量-1)]。
- 调用aclrtQueryDeviceStatus接口查询Device状态是正常可用、还是异常不可用。
- 调用aclrtGetDeviceUtilizationRate接口查询Device上Cube、Vector、AI CPU等的利用率。
- 调用aclrtDeviceGetStreamPriorityRange接口查询硬件支持的Stream最小、最大优先级。
- 调用aclrtGetDeviceInfo接口获取指定Device的AI CPU数量、AI Core数量等信息。

CANN

Context管理

Context作为一个容器，管理了所有对象（包括Stream、Event等）的生命周期。不同Context的Stream、Event是**完全隔离**的，**无法建立同步等待关系**

```
int deviceId = 0;
aclrtContext context = nullptr;
// 初始化 ACL
aclInit(nullptr);
// 设置设备，会隐式创建Defalue Context
aclrtSetDevice(deviceId);

// 显式创建 Context
aclrtCreateContext(&context, deviceId);

// 切换当前 Context（多线程 / 多 Context 场景）
// 设置当前线程使用的 Context
aclrtSetCurrentContext(context);

// 后续的stream、算子执行
// 都发生在该 Context 对应的设备上

// 销毁 Context
aclrtDestroyContext(context);

// 释放设备
aclrtResetDevice(deviceId);
// 反初始化
ACLaclFinalize();
```

CANN

内存管理

Device内存使用

在昇腾异构计算架构中，系统由主机（Host）和设备（Device）组成，Host和Device都有各自独立的内存。为此Runtime提供aclrtMalloc, aclrtFree等接口管理设备内存。

Host锁页内存申请

通常malloc()分配的主机内存是可分页的（pageable）会有如下行为：

- 操作系统可以随时把内存页换出到磁盘
- 物理地址不稳定，只是虚拟地址连续

相对而言，NPU 在进行 DMA（Direct Memory Access）传输时有如下要求：

- 必须使用物理地址稳定的内存
- 不能在传输过程中被 OS 换页。为此Runtime提供Host锁页内存（一般称为Pinned Memory或Page-Locked Memory）功能接口aclrtMallocHost，aclrtFreeHost

```
// 申请host锁页内存
aclrtMallocHost((void**)&hostPtr, size)

// 申请device内存
aclrtMalloc(&devPtr, size, ACL_MEM_MALLOC_HUGE_FIRST);

// H2D内存拷贝
aclrtMemcpy(devPtr, size, hostPtr, size,
ACL_MEMCPY_HOST_TO_DEVICE);

// 释放device内存
aclrtFree(devPtr);
// 释放host内存
aclrtFreeHost(hostPtr);
```

CANN

Stream管理

Stream创建与销毁

```
int32_t deviceId = 0;
// 指定运算的Device
aclrtSetDevice(deviceId);

// 显式创建一个Stream
aclrtStream stream;
aclrtCreateStream(&stream);

// ......

// Stream使用结束后，显式销毁Stream
aclrtDestroyStream(stream);
```

流同步：aclrtSynchronizeStream

```
// 显式创建一个Stream
aclrtStream stream;
aclrtCreateStream(&stream);

// 调用触发任务的接口，传入stream参数
aclrtMemcpyAsync(dstPtr, dstSize, srcPtr, srcSize,
ACL_MEMCPY_HOST_TO_DEVICE, stream);

// 调用aclrtSynchronizeStream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
aclrtSynchronizeStream(stream);

// Stream使用结束后，显式销毁Stream
aclrtDestroyStream(stream);

// ......
```

CANN

Kernel执行

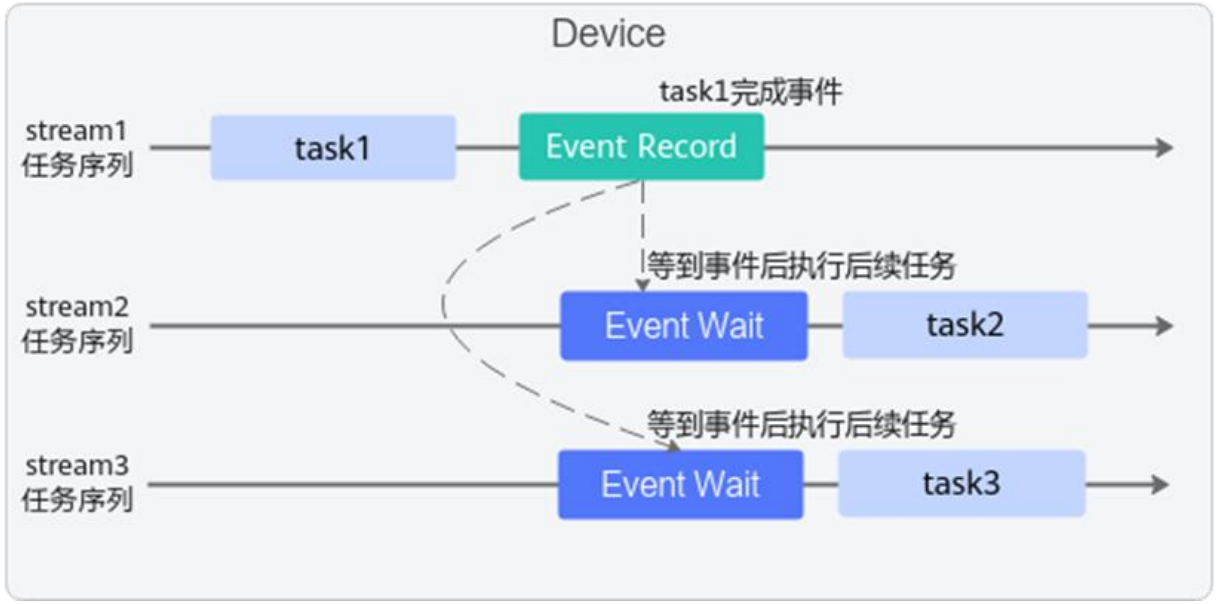
Kernel 加载与执行功能是连接算子实现与硬件执行的核心运行时能力。CANN 支持多种 Kernel 形态（如 Ascend C、自定义算子、TBE 生成算子），并通过统一的 Runtime 接口完成加载、调度和执行

```
const int deviceId = 0;
const int N = 1024;
size_t size = N * sizeof(float);
// 1. 初始化 Runtime
aclInit(nullptr);
aclrtSetDevice(deviceId);
// 2. 申请 Device 内存
float *dA, *dB, *dC;
aclrtMalloc((void**)&dA, size, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc((void**)&dB, size, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc((void**)&dC, size, ACL_MEM_MALLOC_NORMAL_ONLY);

// 3. <<< >>> 方式启动 Kernel
AddKernel<<<blockdim, stream>>>(dA, dB, dC, N);

// 4. 同步
aclrtSynchronizeDevice();
// 5. 资源释放
aclrtFree(dA);
aclrtFree(dB);
aclrtFree(dC);
aclrtResetDevice(deviceId);
aclFinalize();
```

CANN

Event管理

Event通常用于一个Device内、两个Stream之间的任务同步。例如，若stream2的任务依赖stream1的任务，想保证stream1中的任务先完成，这时可创建一个Event，将Event插入到stream1中（通常称为Event Record任务，调用aclrtRecordEvent接口），在stream2中插入一个等待Event完成的任务（通常称为Event Wait任务，调用aclrtStreamWaitEvent接口）。



Event的创建与销毁

```
// 创建Event
aclrtEvent event;
aclrtCreateEvent(&event);
.....
// 销毁Event
aclrtDestroyEvent(event);
```

记录Event时间戳

```
uint64_t time = 0;
float useTime = 0;
// 创建Stream
aclrtStream stream;
aclrtCreateStream(&stream);
// 插入startEvent
aclrtRecordEvent(startEvent, stream);
// 在Stream中下发计算任务
.....
// 插入endEvent
aclrtRecordEvent(endEvent, stream);
aclrtSynchronizeStream(stream);
// 获取时间戳以及计算耗时
aclrtEventGetTimestamp(startEvent, &time);
aclrtEventGetTimestamp(endEvent, &time);
aclrtEventElapsedTime(&useTime, startEvent, endEvent);
```

CANN运行时开源仓访问地址

运行时开源仓地址：
https://gitcode.com/cann/runtime

API文档：
https://gitcode.com/cann/runtime/tree/master/docs

Sample：
https://gitcode.com/cann/runtime/tree/master/example

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