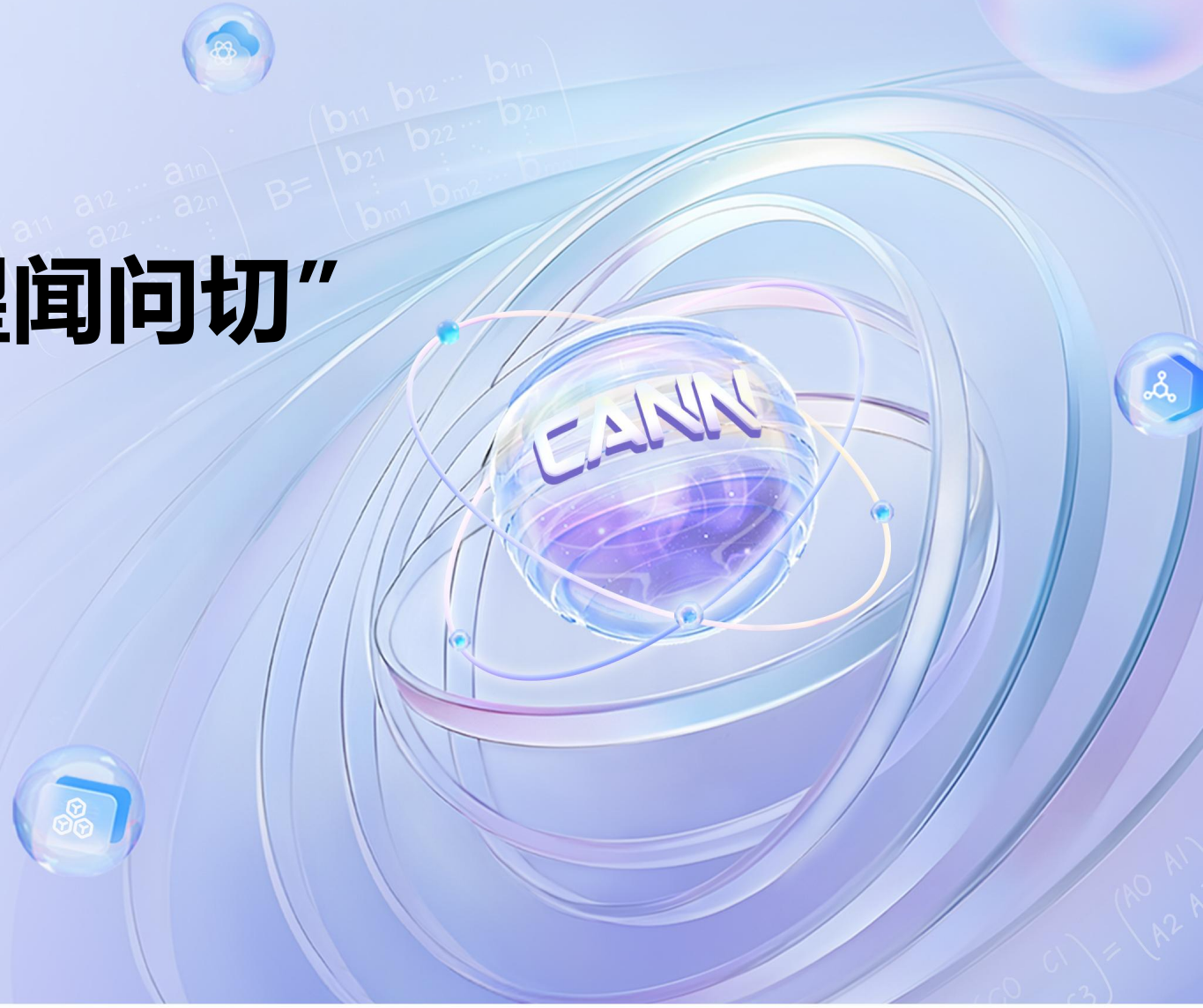
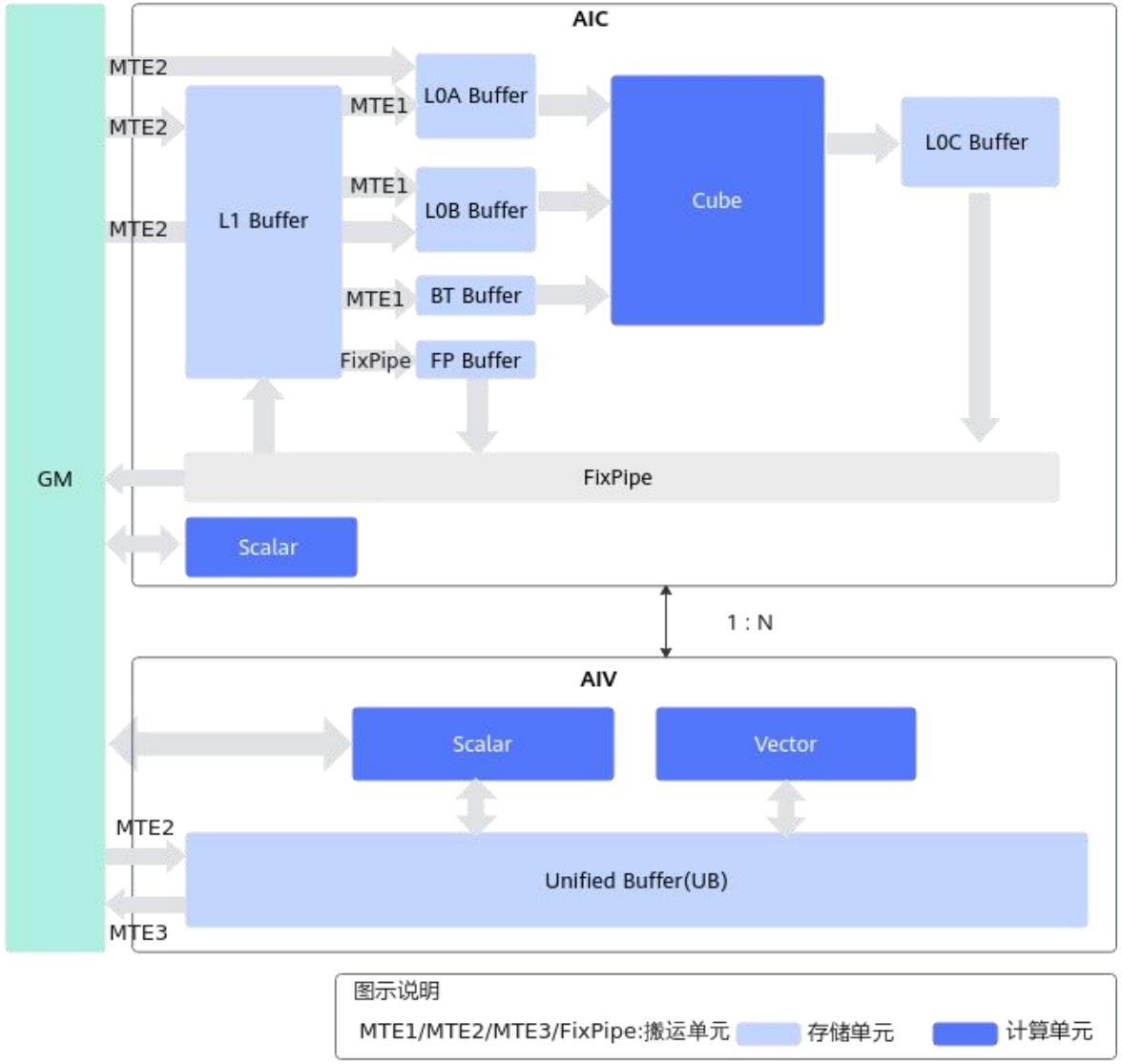
算子性能分析的“望闻问切”



CANN

认识硬件 910B MatMul典型流程



- **硬件架构**
  - 并行执行单元（搬运单元和计算单元）：MTE2, MTE1, FixPipe, Cube，Scalar
  - 多个内存单元：Global Memory ，L1，L0A，L0B，L0C，BT Buffer(Bias) 等
- **典型Cube计算计算数据流**
  - GM->L1->L0A/L0B->[Cube]->L0C->FixPipe->GM

Cube计算单元负责执行矩阵运算，一个核可以在1个时钟周期处理完成16*16*16的矩阵乘法FP16数据类型的矩阵乘

Fp16算力理论值计算方式：

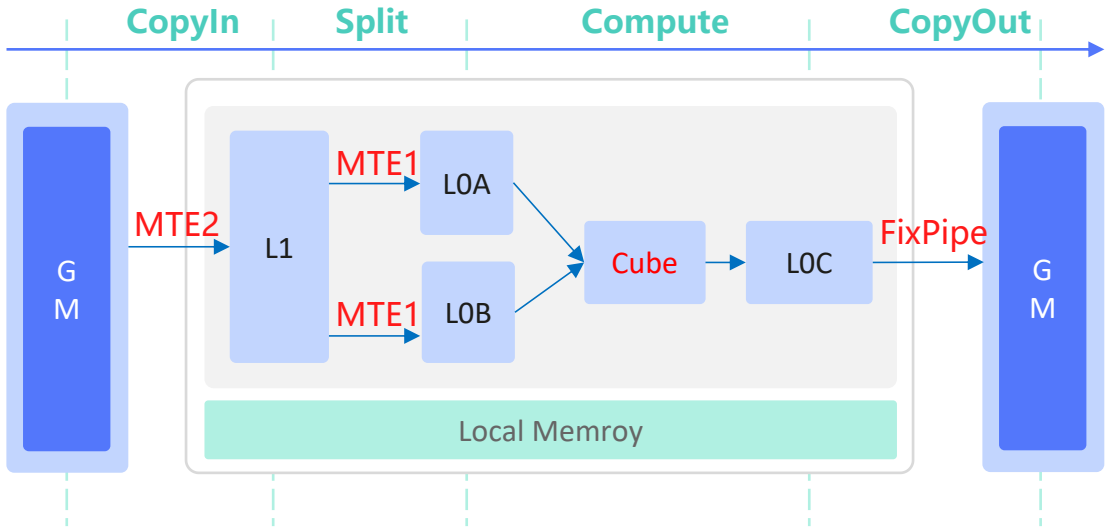
cube：16 * 16* 16 * 频率 * AI核数量 * 2

MatMul算子实现

算子规格描述

算子实现自定义Kernel支持shape： M = 32， N = 32, K=32

	Name	shape	Data type	format
算子输入	a	M * K	Float16	ND
	b	K * N	Float16	ND
算子输出	c	M * N	Float16	ND



GM->L1->L0A/L0B->[Cube]->L0C->FixPipe->GM

执行单元

CopyIn
搬运A/B矩阵到Local内存L1
执行单元： MTE2

DataCopy(a1Local, aGM, nd2nzA1Params);
DataCopy(b1Local, bGM, nd2nzB1Params);

Split
搬运A L1到L0A;
搬运B L1到L0B;
执行单元： MTE1

LoadData(a2Local, a1Local, loadDataParams);
LoadData(b2Local, b1Local, loadDataParams);

Compute
L0A、L0B数据做矩阵乘法
计算结果放到L0C
执行单元： Cube

Mmad(c1Local, a2Local, b2Local, mmadParams);

CopyOut
将结果C矩阵搬出GM
执行单元： FixPipe

Fixpipe(cGM, c1Local, fixpipeParams);

https://www.hiascend.com/document/detail/zh/canncommercial/800/developmentguide/opdevg/Ascendcopdevg/atlas_ascendc_10_00006.html

CANN

性能瓶颈在哪里？—— 性能分析的“望闻问切”

我们的“诊断工具”

工具名称	详细描述
msProf op	采集和分析运行在算子的关键性能指标以快速定位算子的软、硬件性能瓶颈

•Profiling数据： 提供硬件计数器（Pipe利用率等），量化性能瓶颈

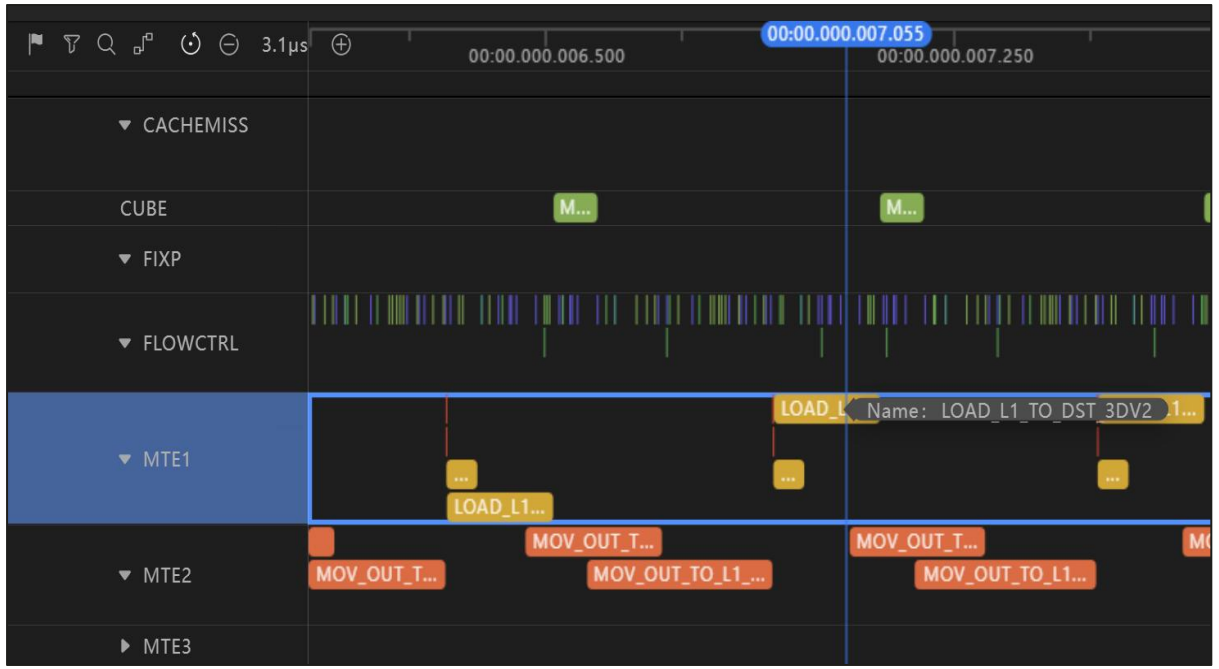
Type	Task Duration(us)	Block Dim	Input Shapes	Output Shapes
Matmul	142.3	20	"2048,7168;1,224,2112,32;2112;2112"	"1,2048,2112"

aic_mac_time(us)	aic_mac_ratio	aic_scalar_time(us)	aic_scalar_ratio
111.812	0.837	43.682	0.327

aic_mte2_time(us)	aic_mte2_ratio	fixpipe_time(us)	fixpipe_ratio
95.77	0.717	9.251	0.069

https://gitcode.com/cann

•指令流水图： 指令流水图以指令维度展示时序关系，并关联调用栈快速定位瓶颈位置



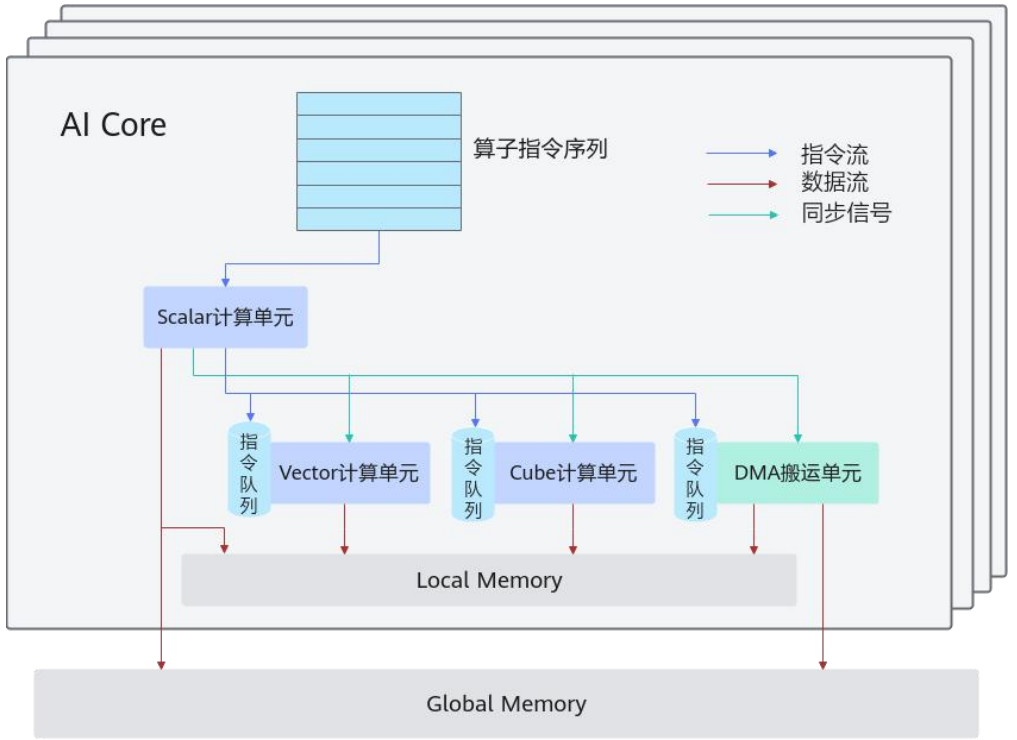
目的：

1. 分析PipeLine利用率（确认输入输出带宽情况）
2. 分析流水并行情况（确认流水并行情况）

https://www.hiascend.com/document/detail/zh/mindstudio/81RC1/ODtools/Operatordevelopmenttools/atlasopdev_16_0082.html

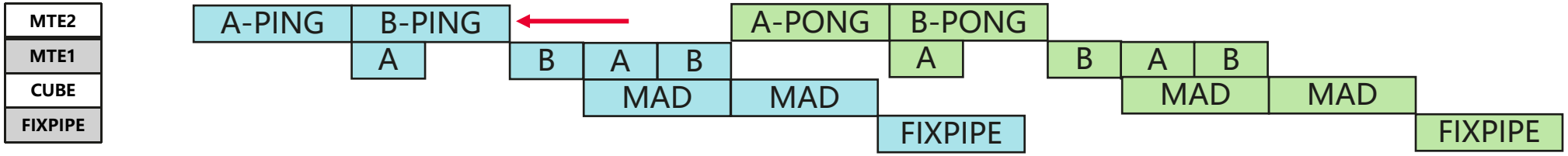
如何系统地进行性能优化？—— 我们的分析框架

基于Profiling与流水图的性能分析流程



流水图优化举例：

下图中各级Pipe看似“有条不紊”，实则各自“摸鱼”，没有真正的各级流水并行起来，为充分利用硬件资源，我们可以做一些尝试……



https://gitcode.com/cann

CANN

算子性能优化案例 – Scalar优化，挖掘“最后百分之一”的潜力

•**问题场景：** 推荐关键矩阵乘法 shape 【700000,4】*【4,4】的Mac利用率仅为4%

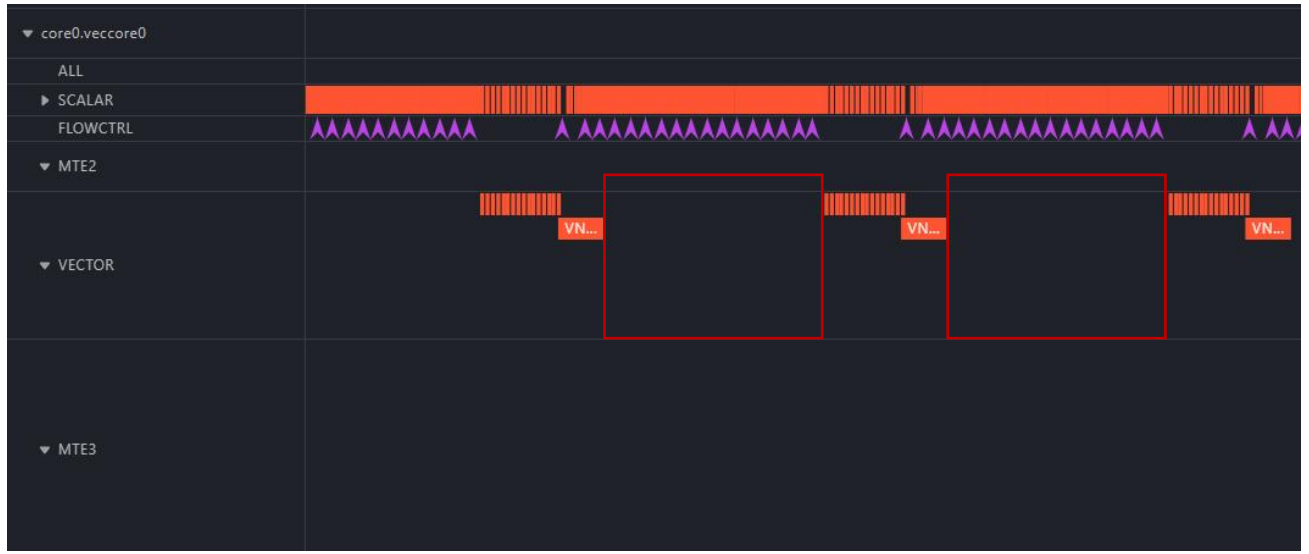
“望闻” – Profiling数据

A	B	C	E	F	G	H
block_id	sub_block_id	aic_time(us)	aic_cube_time(us	aic_cube_ratio	aic_scalar_time(u	aic_scalar_ratio
0	cube0	146.79567	6.604324	0.044029	125.945404	0.839636

K	L	M	N	O	P
aic_mte2_time(us	aic_mte2_ratio	aic_mte3_time(us	aic_mte3_ratio	aic_fixpipe_time((	aic_fixpipe_ratio
28.702162	0.191348	0.771892	0.005146	51.043243	0.340288

Pipe	Time(us)	ratio
Mac	6.6	4%
MTE2	28.7	19%
Fixpipe	51	34%
scalar	125.9	84%

“问切” – 详细分析流水图

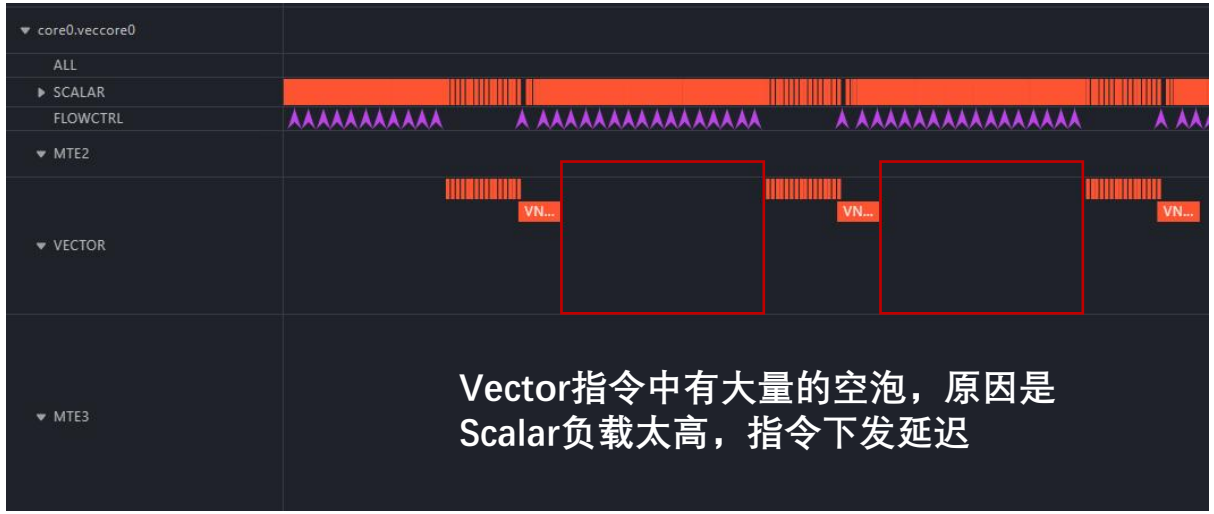


发现Vec指令中出现了大量“空泡”

CANN

算子性能优化案例 – Scalar优化，挖掘“最后百分之一”的潜力

- **现象结论：** 没有任何Pipe超过90%，流水图分析发现Vec指令未发射
- **原因分析：** 仍有部分边缘逻辑、循环控制等由效率较低的Scalar单元处理，成为拖累整体的“短板”。



根因分解：

指令中有大量寄存器配置操作，Scalar负载太高，阻塞指令下发，流水线中大部分时间都在等待

优化方案：

1.识别代码时序依赖关系，计算顺序解耦，公共部分代码提到循环外面

2.开辟栈空间缓存中间计算结果，减少指令下发前的准备动作开销

3.核间同步之前尽量多的执行Scalar操作，减少核唤醒后的Scalar开销

CANN

算子性能优化案例 – 优化方案详解

优化方案：外提Scalar 计算、栈空间储存中间变量，优化了Scalar负载

```
template <class T>
__aicore__ inline void PadDMain(...) {
    ...
    for (uint32_t i = 0; i < VNCHW_SIZE; i++) {
        SetParamForTransData(i, ...);
        TransDataTo5HD(i, ...);
    }
    ...
}
```

```
template <class T>
__aicore__ inline void PadDMain(...) {
    ...
    uint64_t localList[VNCHW_SIZE];  // 申请栈空间，提前计算并保存与i无关的参数
    PrepareParamForTransData(localList, ...);  // 参数设置拆分成两部分：1.与i无关的提前计算并保存；2.与i相关的循环中计算
    for (uint32_t i = 0; i < VNCHW_SIZE; i++) {
        SimpliedSetParamForTransData(i, localList, ...);  // 设置参数的函数被简化，消减指令下发开销导致的阻塞
        TransDataTo5HD(i, ...);
    }
    ...
}
```

优化方案：Scalar前提至核间同步前以减少唤醒开销

```
CUBE:
    CrossCoreWaitFlag(V_NOTIFY_C + pingpong_gm);
    for (int i : task_loop) {
        PrepareForMte2(...);  // 可能会导致MTE2指令下发延迟
        DataCopy(...);
        ...
    }
    CrossCoreSetFlag<0x2, PIPE_MTE2>(C_NOTIFY_V + pingpong_gm);
VECT:
    CrossCoreWaitFlag(C_NOTIFY_V + pingpong_gm);
    for (int i : task_loop) {
        Nd2nzVnchwMM(...)
    }
    CrossCoreSetFlag<0x2, PIPE_MTE3>(V_NOTIFY_C + pingpong_gm);
```

```
CUBE:
    for (int i : task_loop) {
        PrepareForMte2(...);
        if (firstFlag)
            CrossCoreWaitFlag(V_NOTIFY_C + pingpong_gm);  // 调整核同步的时机，尽量多的做完scalar再沉睡
        DataCopy(...);
        if (lastFlag)
            CrossCoreSetFlag<0x2, PIPE_MTE2>(C_NOTIFY_V + pingpong_gm);  // 达成依赖关系后，尽早唤醒
        ...
    }
VECT:
    CrossCoreWaitFlag(C_NOTIFY_V + pingpong_gm);
    for (int i : task_loop) {
        Nd2nzVnchwMM(...)
    }
    CrossCoreSetFlag<0x2, PIPE_MTE3>(V_NOTIFY_C + pingpong_gm);
```


性能提升：推荐领域大M小KN Scalar Bound场景**性能优化10%左右。**

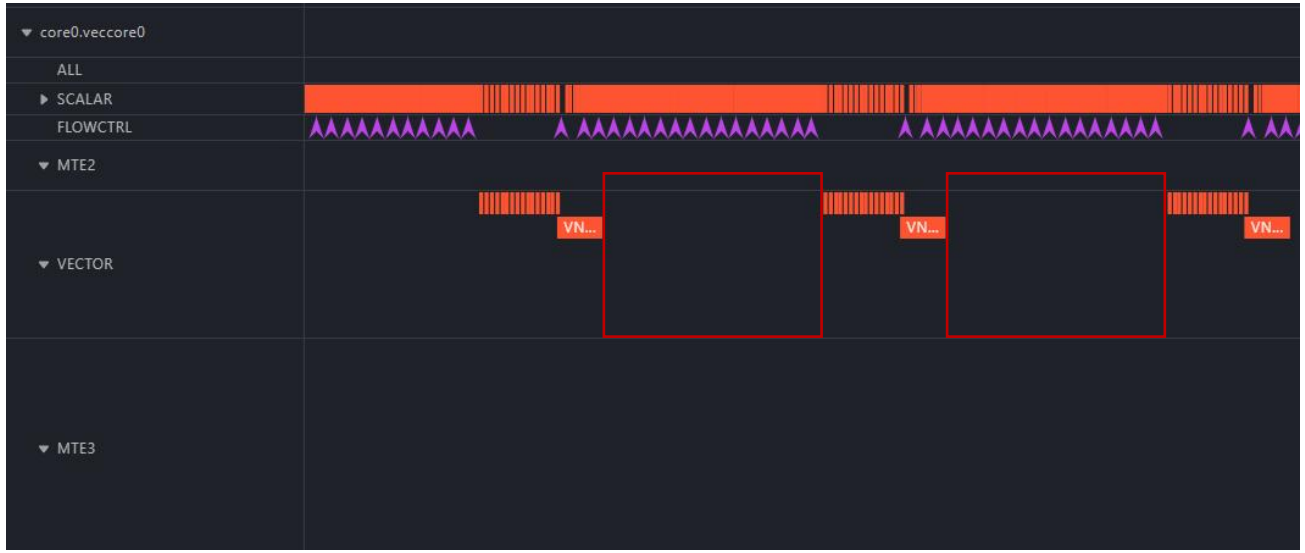
CANN

算子性能优化案例 – 优化方案效果展示

优化前（Pipe“偷懒”）：

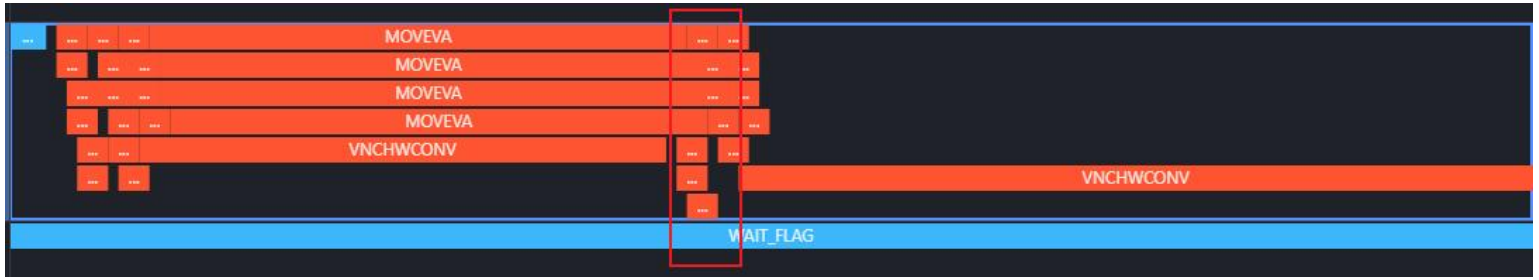
Scalar负载太高阻塞指令下发，流水线中大部分时间都在等待

硬件资源无法充分发挥



优化后（Pipe“干劲满满”）：

Vec流水指令几乎排满，流水间“空泡”消失，硬件能力得到充分发挥



CANN

总结与展望——共赴性能优化之旅

方法论回顾

- 数据驱动：从Profiling数据分析而非猜测
- 识别瓶颈：目标是硬件资源的充分发挥
- 分层优化：从架构到指令，层层深入
- 实现验证：反复迭代，滚动前进

基于Profiling与流水图的性能分析流程

影响算子性能的四大关键“症结”

序号	观测点	可能瓶颈点	影响因素	可能优化手段
1	Mac Ratio 是多少?	算力瓶颈	算力	认为充分发挥算力
2	MTE2 Ratio 是多少?	输入带宽瓶颈	输入带宽	减少重复搬运、离线亲和格式、L2 Cache命中率优化
3	Fixpipe Ratio 是多少?	输出带宽瓶颈	输出带宽	调整搬出基本块大小、私有格式搬出优化
4	没有任何Pipe利用率高	流水并行度不足	软件实现	数据搬运流水优化、硬件单元间的流水设计、计算和搬运并行

CANN

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