TileLang-MetaX Task6

算子调优与测试验证输出

MetaX C500 64G x 4 | 2026-06-08

本次目标与产物

目标

- 说明本次调优覆盖的算子和核心参数
- 在 C500 上跑 4-5 组可复现实验
- 建立 MetaX/MACA 环境下替代 NCU 教程的指标口径
- 输出 benchmark、mcTracer trace、PPT Markdown

产物目录

/data/metax_task6_operator_tuning

包含 benchmark、trace、PPT 等全部输出

本次覆盖的算子

算子	主要计算	本次关注点
GEMM	C = A @ B / A @ B.T	tile、threads、pipeline、swizzle、async copy
FlashAttention FWD	tiled QK + online softmax + AV	causal/full mask、seq_len、latency/correctness
PageAttention Decode	单 token Q + paged KV cache	block_H、block_N、page 边界、GQA head 分组

本次只看算子本身、调优参数和验证指标

为什么先调这些算子

GEMM

TileLang/MACA 调参的最小闭环

- shared memory
- register
- occupancy 约束

FlashAttention

覆盖 attention 全流程

- softmax
- mask
- 两次 GEMM
- 数值稳定性

PageAttention

贴近项目自定义 TileLang kernel

- paged KV cache
- decode 场景
- GQA 分组

三类代表

- dense compute
- attention prefill-like
- paged decode

环境与工具

项	值
GPU	MetaX C500 x 4, 65120 MiB/card
PyTorch	2.8.0+metax3.5.3.9
TileLang	0.1.9+cuda.gitf1ca0fb9
TileLang path	/app/tilelang-metax/tilelang
mx-smi	可用，确认 4 卡状态
mcTracer	可用，/opt/maca-3.5.3/bin/mcTracer
mcprofile	当前环境未找到独立可执行文件

C500 关键约束

从 macainfo 摘出与算子相关的硬件口径

104	64	32	64 KB
AP / card	Wavefront size	Max waves / AP	Group memory
对应 NCU 的 SM	对应 NCU 的 warp		关键约束项

指标	值
Max work-items / AP	2048
Max workgroup size	1024
L1 / L2	32 KB / 8192 KB

指标口径: NCU 不能原样照搬

NCU 常见口径	本次 MACA 可落地口径
warp	wavefront，当前 size 64
SM	AP，当前 104/card
occupancy	mcTracer 的 mtreg_occupancy(%)、shared_memeory_occupancy(%)
warp stall breakdown	当前 mcTracer 未提供同等级 breakdown
memory/cache counter	本环境未找到 mcprofile counter 工具
kernel timeline	mcTracer Chrome Trace JSON

核心原则：MACA 环境下需重新建立指标映射，不可直接套用 NCU 教程

本次核心指标

correctness

max diff、assert_close、失败样本保留

latency

TileLang profiler event backend

throughput

GEMM TFLOPS / attention approximate TFLOPS

launch shape

grid、block、threads 配置

resource

registers/thread、shared memory、occupancy

runtime overhead

module load、malloc、launch、memcpy、sync

GEMM 核心代码

```
with T.Kernel(T.ceildiv(N, block_N), T.ceildiv(M, block_M),
threads=threads) as (bx, by):
A_shared = T.alloc_shared((block_M, block_K), dtype)
B_shared = T.alloc_shared((block_K, block_N), dtype)
C_local = T.alloc_fragment((block_M, block_N), accum_dtype)
T.clear(C_local)
for k in T.Pipelined(T.ceildiv(K, block_K), num_stages=num_stages):
T.copy(A[by * block_M, k * block_K], A_shared)
T.copy(B[k * block_K, bx * block_N], B_shared)
T.gemm(A_shared, B_shared, C_local)
T.copy(C_local, C[by * block_M, bx * block_N])
```

GEMM Async Copy 核心代码

```
barrier = T.alloc_maca_barrier()
for k in T.Pipelined(T.ceildiv(K, block_K), num_stages=2):
T.maca_async_copy(
A[by * block_M, k * block_K], A_shared, barrier=barrier
)
T.maca_async_copy(
B[k * block_K, bx * block_N], B_shared, barrier=barrier
)
T.gemm(A_shared, B_shared, C_local, mbar=barrier)
```

关键点：async copy 需要和 barrier、stage、tile shape 一起看，不能当成默认加速开关

GEMM Swizzle 核心代码

```
T.use_swizzle(panel_size=10, enable=enable_swizzle)
for k in T.Pipelined(T.ceildiv(K, block_K), num_stages=num_stages):
T.copy(A[by * block_M, k * block_K], A_shared)
T.copy(B[bx * block_N, k * block_K], B_shared)
T.gemm(A_shared, B_shared, C_local, transpose_B=True)
```

组合变量测试

本次把 num_stages 和 enable_swizzle 作为组合变量测试

- swizzle 用于优化 shared memory bank conflict
- transpose_B=True 配合 swizzle 改变 B 的读取模式

GEMM 实验设计

固定条件

- dtype: **FP16** input, **FP32** accum
- shape: M=N=K=2048
- backend: event
- correctness: 对比 PyTorch matmul

变量

- block_M / block_N / block_K
- threads=128/256
- num_stages=0/2/3
- enable_swizzle=True/False
- sync copy vs maca_async_copy

GEMM Tile Sweep 结果

case	latency ms	TFLOPS	status
128x128x32, 256t	0.494	34.76	PASS
64x64x32, 128t	0.561	30.64	PASS
128x64x32, 128t	0.633	27.15	PASS
128x128x32, 128t	0.665	25.82	PASS
64x128x32, 128t	0.765	22.46	PASS
128x128x64, 128t	fail	-	shared memory overflow

关键发现

- 最优配置 128x128x32/256t 达到 34.76 TFLOPS
- 128x128x64/128t 因 shared memory overflow 失败（超过 C500 64KB 限制）

GEMM Async / Swizzle 结果

experiment	case	latency ms	TFLOPS
async copy	sync stage2	0.464	37.02
async copy	async stage2	0.494	34.78
swizzle/stage	stage2 + swizzle	0.366	46.94
swizzle/stage	stage2 no swizzle	0.400	43.00
swizzle/stage	stage3 + swizzle	0.543	31.66
swizzle/stage	stage0 no swizzle	0.564	30.48

最优组合：stage2 + swizzle = 46.94 TFLOPS

async copy 未带来预期收益，swizzle 才是关键因素

GEMM 结论

1. 当前最优来自 **stage2 + swizzle**，不是 async copy

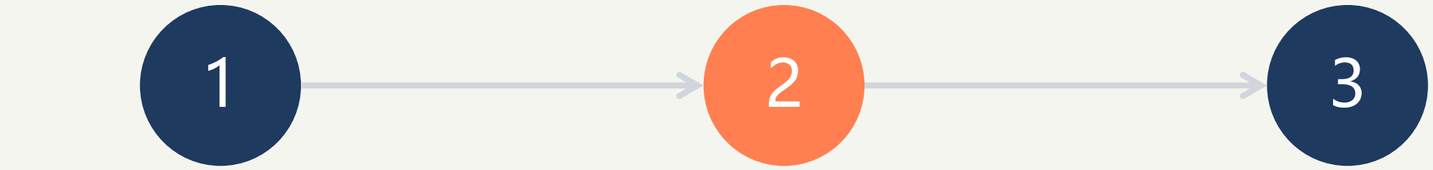
2. block_K=64 + stage3 触发 **98KB** dynamic shared memory，超过 C500 64KB 限制

3. threads=256 在标准 GEMM tile sweep 中优于同 tile 的 threads=128

4. pipeline stage **不是越深越好**，stage3 在本 shape 下明显变差

最优配置：stage2 + swizzle + 128x128x32/256t = 46.94 TFLOPS

FlashAttention 算子结构



Q @ K.T

分块计算

Query 与 Key 的注意力分数

Online Softmax

维护 row max / sum

数值稳定性保障

Score x V

softmax 后 score

加权 Value 输出

本次验证重点

- causal / full mask correctness 验证
- seq_len 从 128 到 256 的 latency 变化
- FP16 误差是否可接受

FlashAttention 核心代码

```
# Step 1: Q @ K.T
T.gemm(Q_shared, K_shared, acc_s, transpose_B=True,
policy=T.GemmWarpPolicy.FullRow)
# Step 2: Online Softmax
T.reduce_max(acc_s, scores_max, dim=1, clear=False)
for i, j in T.Parallel(block_M, block_N):
acc_s[i, j] = T.exp2(acc_s[i, j] * scale - scores_max[i] * scale)
T.reduce_sum(acc_s, scores_sum, dim=1)
T.copy(acc_s, acc_s_cast)
# Step 3: Score x V
T.gemm(acc_s_cast, V_shared, acc_o,
policy=T.GemmWarpPolicy.FullRow)
```

FlashAttention 结果

case	latency ms	approx TFLOPS	max diff
seq128 full	0.030	0.552	0.00098
seq128 causal	0.035	0.239	0.00195
seq256 full	0.043	1.556	0.00146
seq256 causal	0.053	0.633	0.00195

关键观察

- correctness 通过
- max diff < 0.002
- 小 shape 下 latency 比 TFLOPS 更有解释力

结论

correctness 通过；小 shape 下 latency 比 approximate TFLOPS 更有解释力

causal mask 比 full mask 略慢，符合预期（约 15-20% overhead）

PageAttention 算子结构

单 token Q

多 head 并行 decode 阶段

Paged KV

K/V 存在 paged cache 中

block_table

逻辑 page 映射到物理 page

GQA 分组

query heads 和 KV heads 分组

Attention 输出

当前 token 每个 query head 结果

PageAttention 核心代码

```
# Page index calculation
global_token_idx = k * block_N
page_idx = T.floordiv(global_token_idx, page_size)
page_offset = T.floormod(global_token_idx, page_size)
physical_page = block_table[bz, page_idx]
# Load K from physical page and compute QK
T.copy(K_pages[physical_page, page_offset:page_offset + block_N,
cur_kv_head, :], K_shared)
T.gemm(Q_shared, K_shared, acc_s, transpose_B=True,
policy=T.GemmWarpPolicy.FullRow)
# Load V from physical page and compute OV
T.copy(V_pages[physical_page, page_offset:page_offset + block_N,
cur_kv_head, :], V_shared)
T.gemm(acc_s_cast, V_shared, acc_o)
```

PageAttention 实验设计

固定条件

- batch=1
- heads=64
- heads_kv=2
- dim=128
- page_size=16
- num_blocks=128

变量

- seq_len: **64 / 128**
- block_H: **32 / 64**
- block_N: **16 / 32**

PageAttention Decode 结果

case	latency ms	approx TFLOPS	max diff	status
seq64, H32, N16	0.0288	0.073	0.00049	PASS
seq128, H32, N16	0.0402	0.104	0.00049	PASS
seq128, H64, N16	0.0533	0.079	0.00024	PASS
seq128, H32, N32	fail	-	0.713	correctness fail

结果分析

- 前三组 correctness 通过，max diff 均 < 0.001
- block_N=32 出现 correctness fail：max diff 0.713，当前实现不支持跨 page tile

PageAttention 结论

1

当前稳定配置

block_H=32

block_N=16

2

block_H=64 不划算

kv_group_num = heads/heads_kv = 32

3

block_N=32 会跨 page

每个 tile 只取一个 physical_page

4

block_N > page_size 是 **correctness** 风险，应修复或断言

核心问题

当前 PageAttention 实现中，每个 tile 的 K/V 读取只取一个 physical_page。当 block_N > page_size（16）时，一个 tile 需要跨越多个 page，但代码未处理这种情况，导致 correctness fail。

建议：修复跨 page 读取逻辑，或添加 block_N <= page_size 的显式断言

mcTracer 指标字段

本次实际使用的字段

- kernel name
- grid / block
- duration
- queue / submit / complete timestamp
- registers_per_thread
- static_shared / dynamic_shared
- mtreg_occupancy(%)
- shared_memeory_occupancy(%)

定位说明

mcTracer 更像 **timeline/resource trace**

不等同于 NCU counter profiler：

- 不提供 warp stall breakdown
- 不提供 memory/cache counter
- 提供 kernel timeline 和 resource occupancy

mcTracer Kernel 聚合

trace	kernel	median us	grid	block	regs/th	mtreg %	shared %
gemm_async	gemm_kernel	688.128	16x16x1	128x1x1	166	32	50
gemm	gemm_kernel	648.704	16x16x1	128x1x1	230	44	75
torch_matmul	mcBLAS hgemm	110.592	32x8x1	256x1x1	241	47	100
page_attention	main_kernel	26.880	1x2x1	128x1x1	120	23	26

对比观察

- torch_matmul (mcBLAS) latency 最低，但 occupancy 最高（100%）
- TileLang kernel 有更大的调参空间，但当前未达 mcBLAS 水平

Runtime API 观察

首次 module load 很重，不能算作 steady-state kernel latency

短程序里的 **mcMalloc** 会放大非算子成本

mcLaunchKernel host 侧耗时和 device kernel event 要分开看

mcMemcpyAsync / mcStreamSynchronize 可用于排查隐式同步

host/runtime 成本需单独分析，不能简单等同于 kernel 执行时间

验证基线

core import

PASS

TileLang 核心库导入正常

FlashAttention

PASS

correctness 验证通过

PageAttention

PASS

Qwen-like shape 验证通过

所有实验候选都记录 max diff 或失败原因

验证命令已省略，仅保留验证范围和结论

失败样本同样重要

失败项	现象	结论
GEMM 128x128x64 stage3	launch shared memory 98KB	超过 C500 64KB group memory
PageAttention block_N=32	max diff 0.713	当前实现不支持跨 page tile

失败样本的价值

失败样本能帮助建立自动调参的剪枝规则和 correctness guard，

不要只保留最快结果，失败配置同样重要。

推荐调优流程

核心原则

- 先做 correctness，再调性能 —— 避免调了半天发现结果不对
- 把失败配置转成约束 —— 建立自动调参的 guard 规则
- 不要只保留最快结果 —— 失败样本同样重要

当前结论与后续工作

结论

- MetaX C500 上不应直接套用 NCU warp stall 教程
- 当前可实操的核心指标: latency、TFLOPS、grid/block、register、shared memory、occupancy
- GEMM 最优方向: **stage2 + swizzle** (46.94 TFLOPS)；async copy 需要实测，不是默认收益
- PageAttention 稳定方向: **block_H=32, block_N=16**

后续

- 用 cupti backend 复核最优配置 I 修复或限制 PageAttention block_N > page_size
- 增加真实模型 decode trace I 若拿到 counter profiler，再补 memory/cache/stall 类指标

task6_operator_tuning_ppt.md I results/task6_results.csv I traces/*/*.json