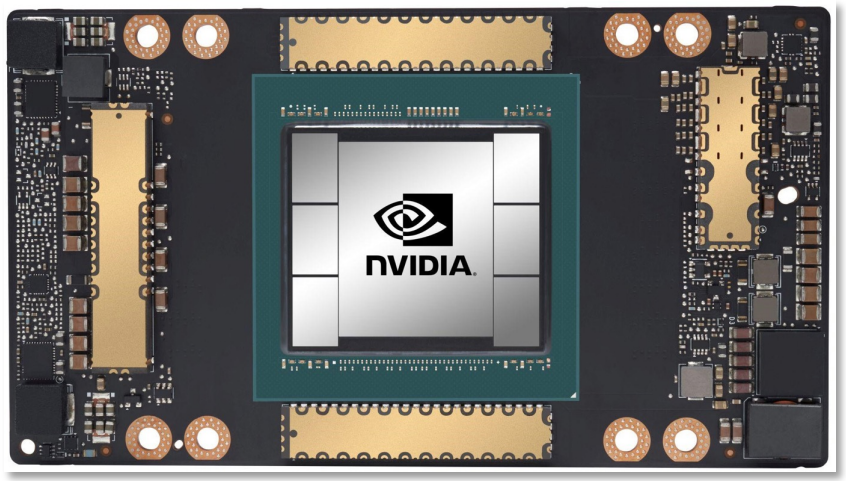
CUDA实践下的 GPU并行计算原理与机制介绍

博士后： 纪焘

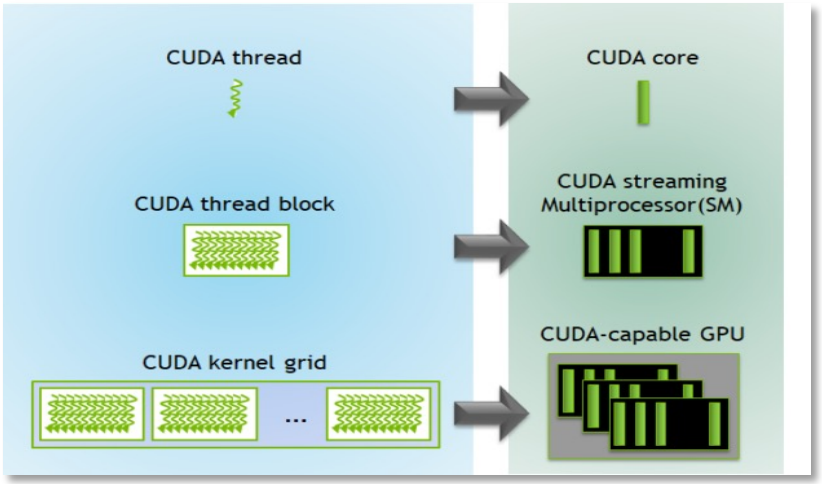
助教： 熊静飞、郭彬

taoji@fudan.edu.cn, jfxiong24@m.fudan.edu.cn, binguo@stu.ecnu.edu.cn

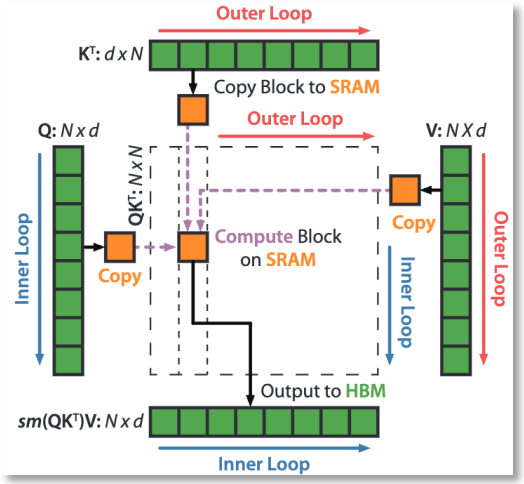
AI-Infra Seminar



GPU



CUDA、Triton、~~Cutlass~~



FlashAttention

Q&A time

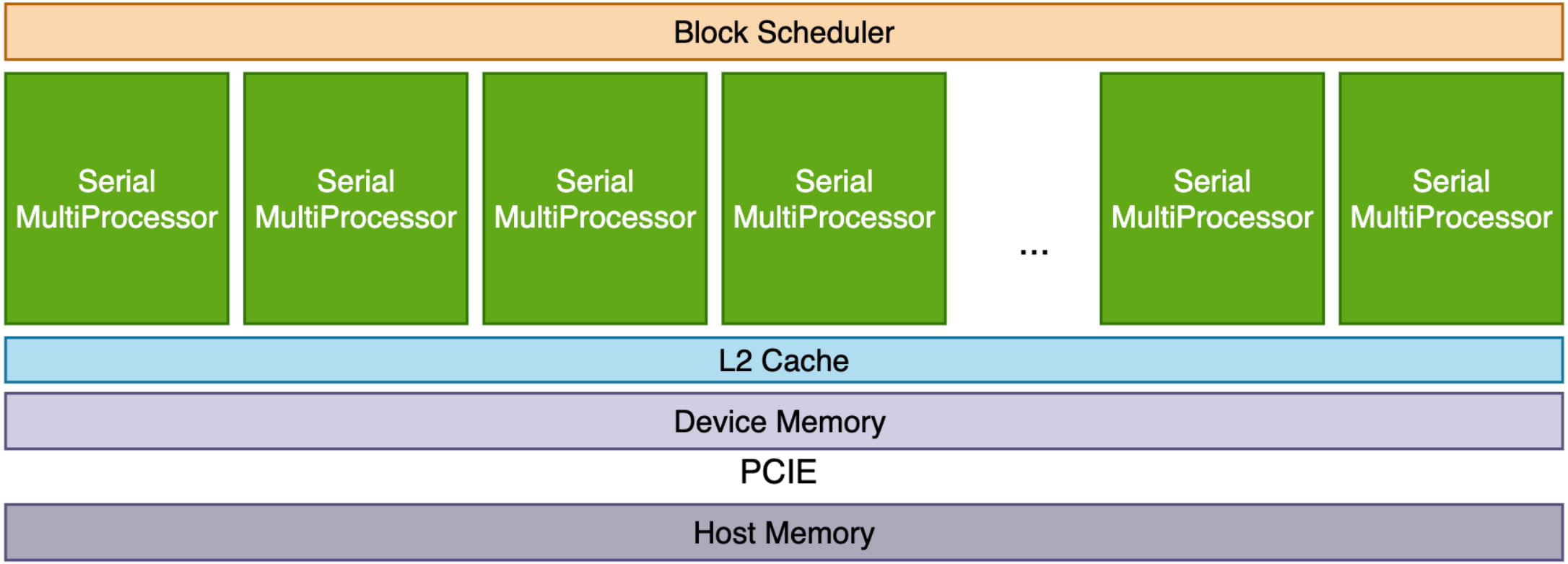
- 17:15 ~ 18:00（250715~250717）
- A5029, 江湾叉二
- #腾讯会议： 354-8238-6983

Lecture

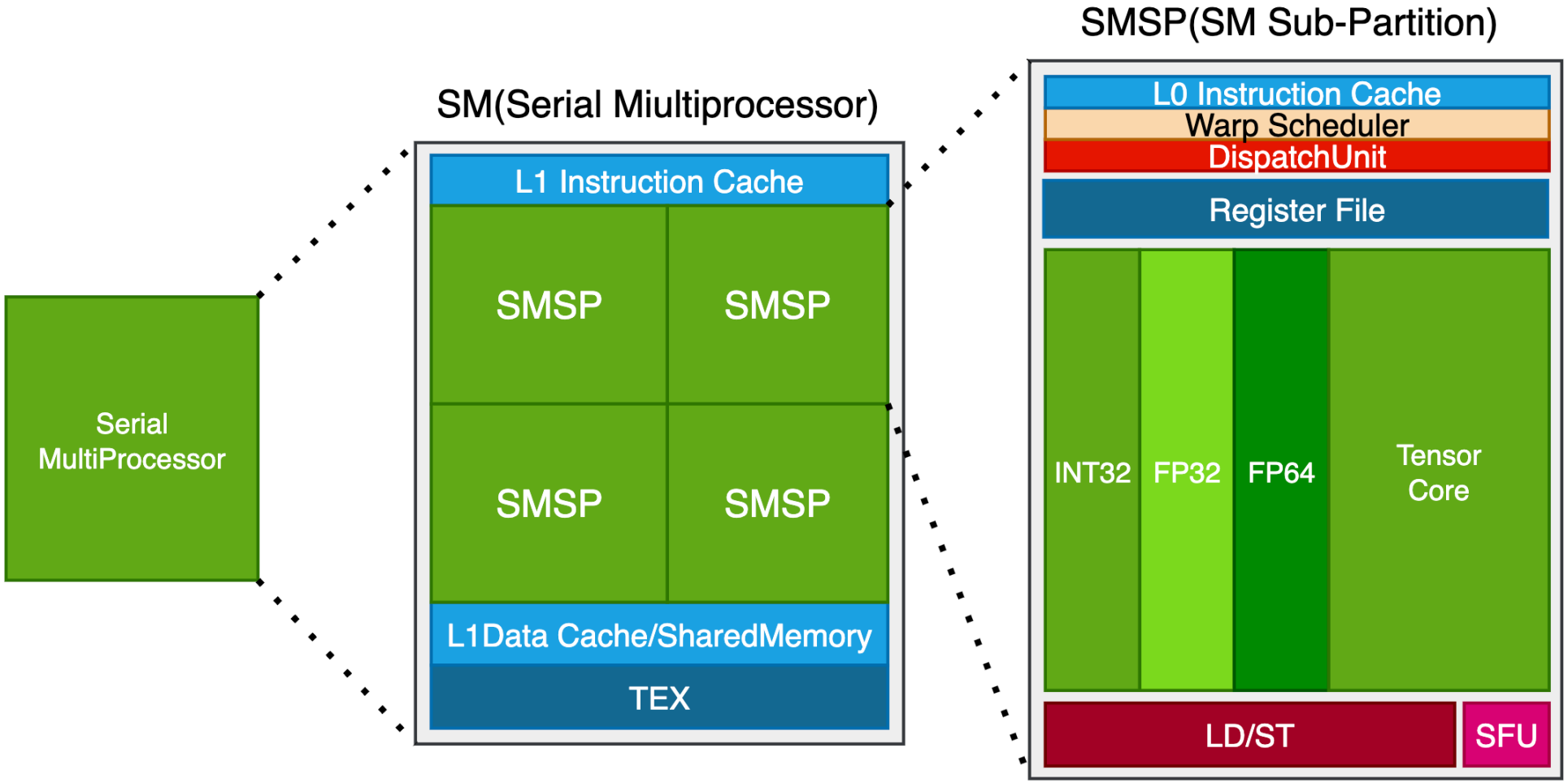
https://github.com/JT-Ushio/AI-Infra-Seminar

群聊：并行计算&CUDA讨论班

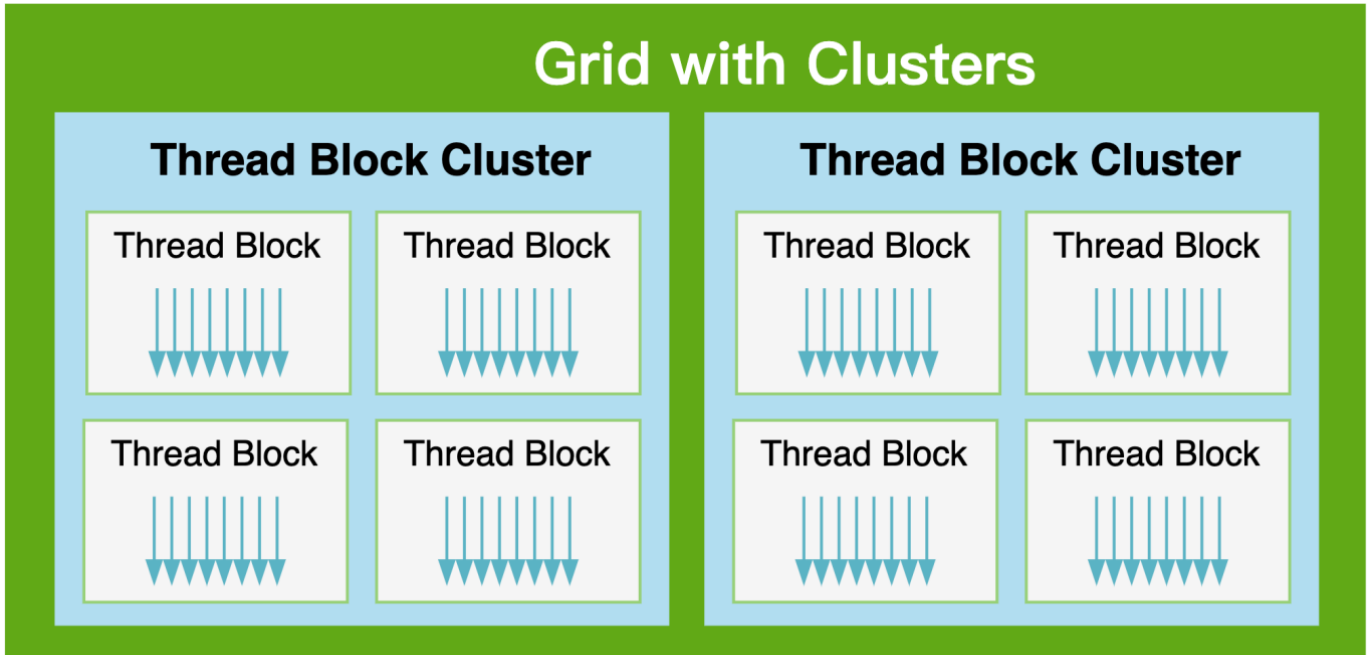
GPU


Block Scheduler
Serial MultiProcessor
Serial MultiProcessor
Serial MultiProcessor
Serial MultiProcessor
...
Serial MultiProcessor
Serial MultiProcessor
L2 Cache
Device Memory
PCIE
Host Memory

GPU


SMSP(SM Sub-Partition)
SM(Serial Miultiprocessor)
Serial
MultiProcessor
L1 Instruction Cache
SMSP
SMSP
SMSP
SMSP
L1Data Cache/SharedMemory
TEX
L0 Instruction Cache
Warp Scheduler
DispatchUnit
Register File
INT32
FP32
FP64
Tensor
Core
LD/ST
SFU

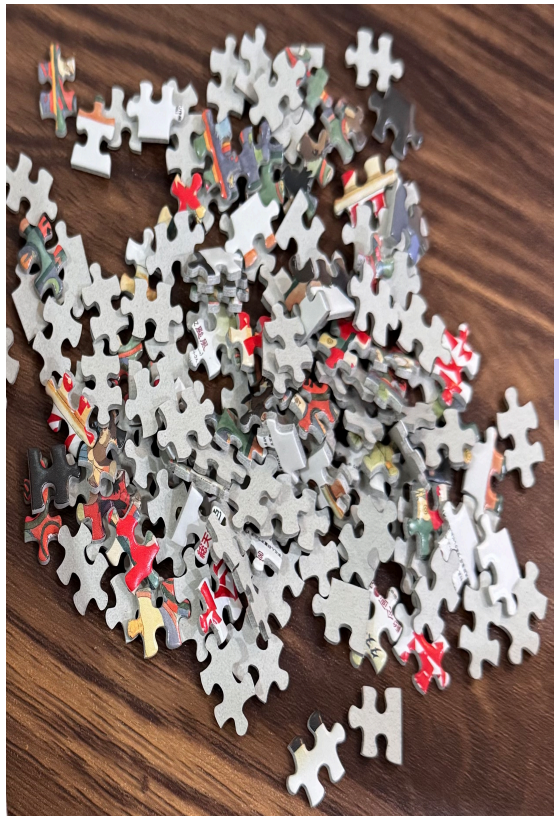
CUDA Programming



software concept	hardware concept
grid(Kernel) 108/114/132SMs	device
thread block cluster (TBC) 12~14SMs	GPU Processing Cluster (GPC)
thread block (CTA) ⩽1024	Serial MultiProcessor(SM) ⩽2048
warp 32*1	Serial MultiProcessor Sub-Partitiom (SMSP)
thread 1	cuda core

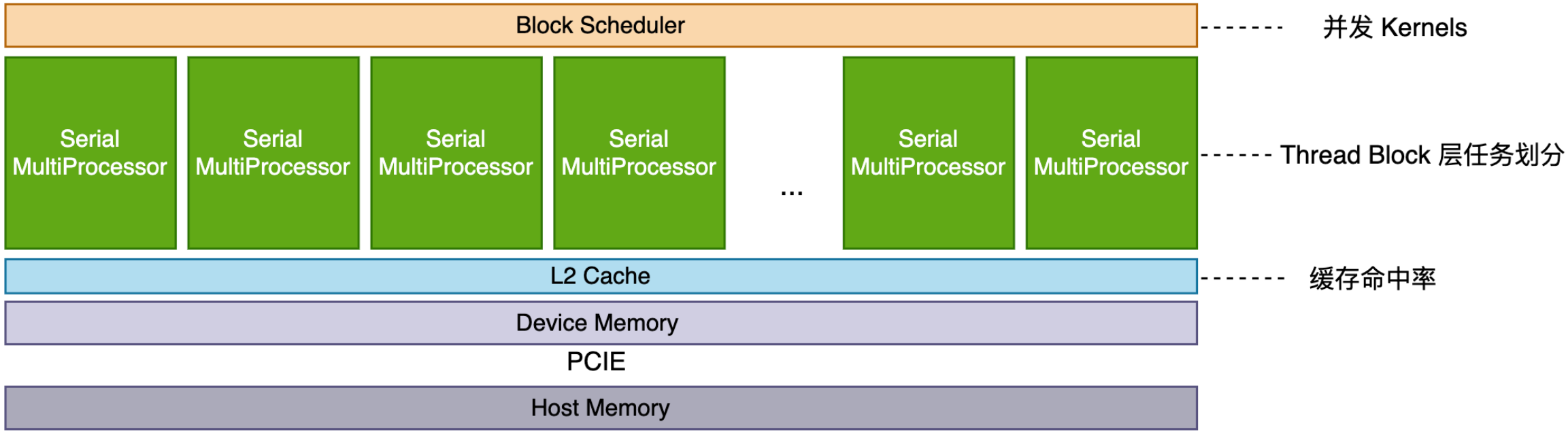
- **Thread-level Programming**
- **Single instruction, multiple threads (SIMT)**

CUDA Programming: SIMT



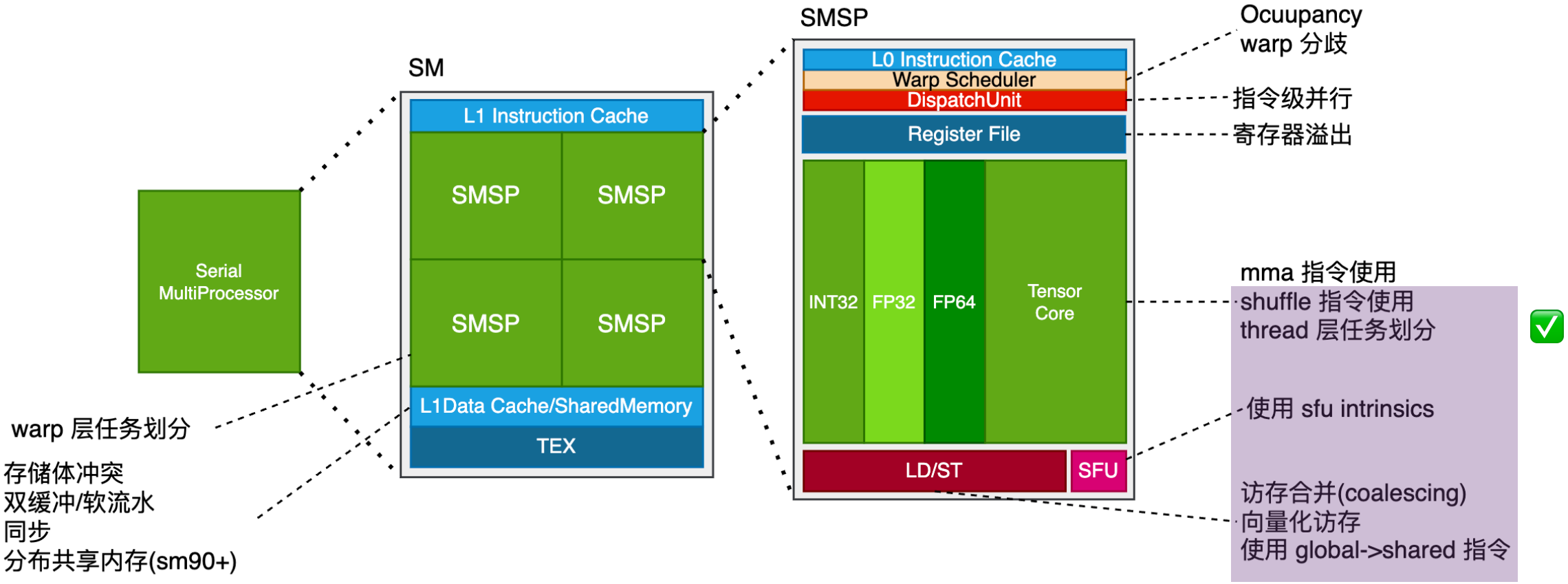
- ID
- R, C=ID // Col, ID % Col
- 🧩

CUDA Programming

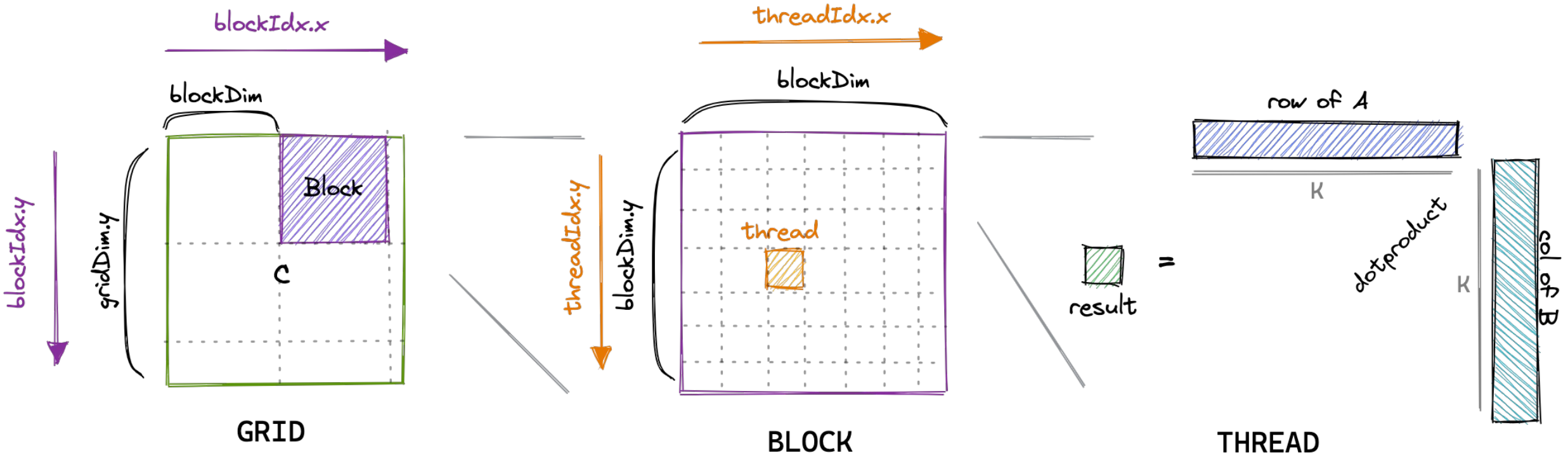


outside an SM

Conclusion


SMSP
Ocuupancy
warp 分歧
SM
L0 Instruction Cache
Warp Scheduler
DispatchUnit
指令级并行
Register File
寄存器溢出
L1 Instruction Cache
SMSP
SMSP
SMSP
SMSP
Serial
MultiProcessor
INT32
FP32
FP64
Tensor
Core
mma 指令使用
shuffle 指令使用
thread 层任务划分
L1Data Cache/SharedMemory
TEX
LD/ST
SFU
使用 sfu intrinsics
warp 层任务划分
存储体冲突
双缓冲/软流水
同步
分布共享内存(sm90+)
访存合并(coalescing)
向量化访存
使用 global->shared 指令

Matmul v0


blockIdx.x
blockDim
blockIdx.y
gridDim.y
Block
C
GRID
We put as many blocks into the grid as necessary to span all of C
threadIdx.x
blockDim
threadIdx.y
blockDim.y
thread
BLOCK
Each block is responsible for calculating a 32x32 chunk of C
result
=
row of A
K
dotproduct
K
col of B
THREAD
Each thread independently computes one entry of C

Matmul v0

0_naive_matmul.py

```python
module = load_inline(
    name="matmul_naive_mod",
    cpp_sources=cpp_source,
    cuda_sources=cuda_source,
    functions=["matmul_naive"],
    verbose=True,
)

M, K, N = 1024, 1024, 1024
A = torch.randn(M, K, device="cuda").contiguous()
B = torch.randn(K, N, device="cuda").contiguous()

with profile(
    activities=[ProfilerActivity.CUDA],
    schedule=torch.profiler.schedule(wait=1, warmup=0, active=2),
    on_trace_ready=torch.profiler.tensorboard_trace_handler("./log/matmul_naive")
,   record_shapes=False,
    with_stack=False,
) as prof:
    for _ in range(3):
        with record_function("matmul_naive"), torch.no_grad():
            C1 = module.matmul_naive(A, B)
        with record_function("torch_matmul"), torch.no_grad():
            C2 = torch.matmul(A, B)
        prof.step()

assert torch.allclose(C1, C2, atol=1e-3, rtol=1e-3)
print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))

```

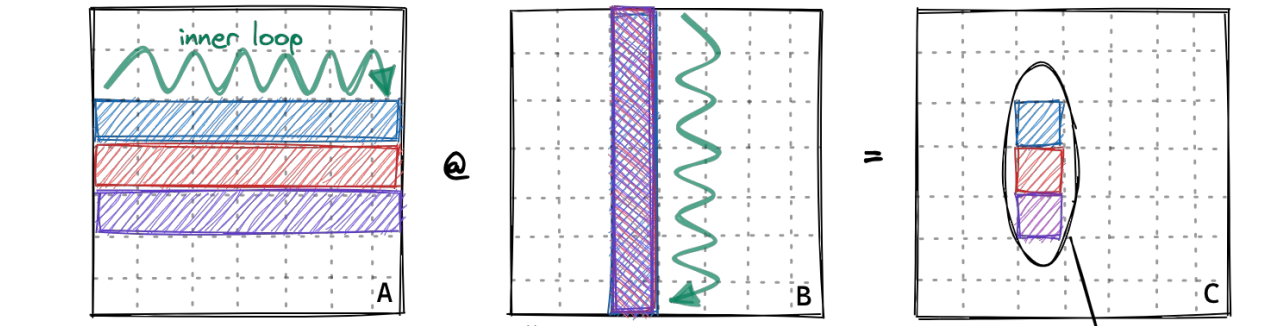
Matmul v0

0_naive_matmul.py

```
cuda_source = """
__global__ void matmul_naive_kernel(const float* A, const float* B, float* C, int M, int N, int K) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < M && col < N) {
        float val = 0.0f;
        for (int k = 0; k < K; ++k) {
            val += A[row * K + k] * B[k * N + col];
        }
        C[row * N + col] = val;
    }
}

torch::Tensor matmul_naive(torch::Tensor A, torch::Tensor B) {
    TORCH_CHECK(A.size(1) == B.size(0), "Incompatible matrix sizes");
    int M = A.size(0), K = A.size(1), N = B.size(1);

    auto C = torch::empty({M, N}, A.options());

    dim3 threads(32, 32);
    dim3 blocks((N + threads.x - 1) / threads.x, (M + threads.y - 1) / threads.y);

    matmul_naive_kernel<<<blocks, threads>>>(
        A.data_ptr<float>(), B.data_ptr<float>(), C.data_ptr<float>(), M, N, K
    );

    return C;
}
"""
```

Matmul v0

Naive kernel:

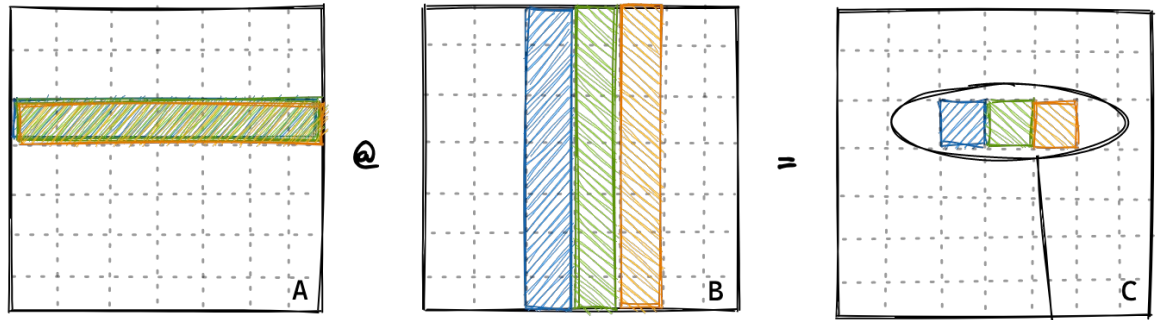


threads access non-consecutive values ⇒ cannot coalesce

all threads access same values ⇒ within-warp broadcast

No benefit to putting these threads in same warp

Coalescing kernel:



all threads access same values ⇒ within-warp broadcast

threads access consecutive values ⇒ can coalesce

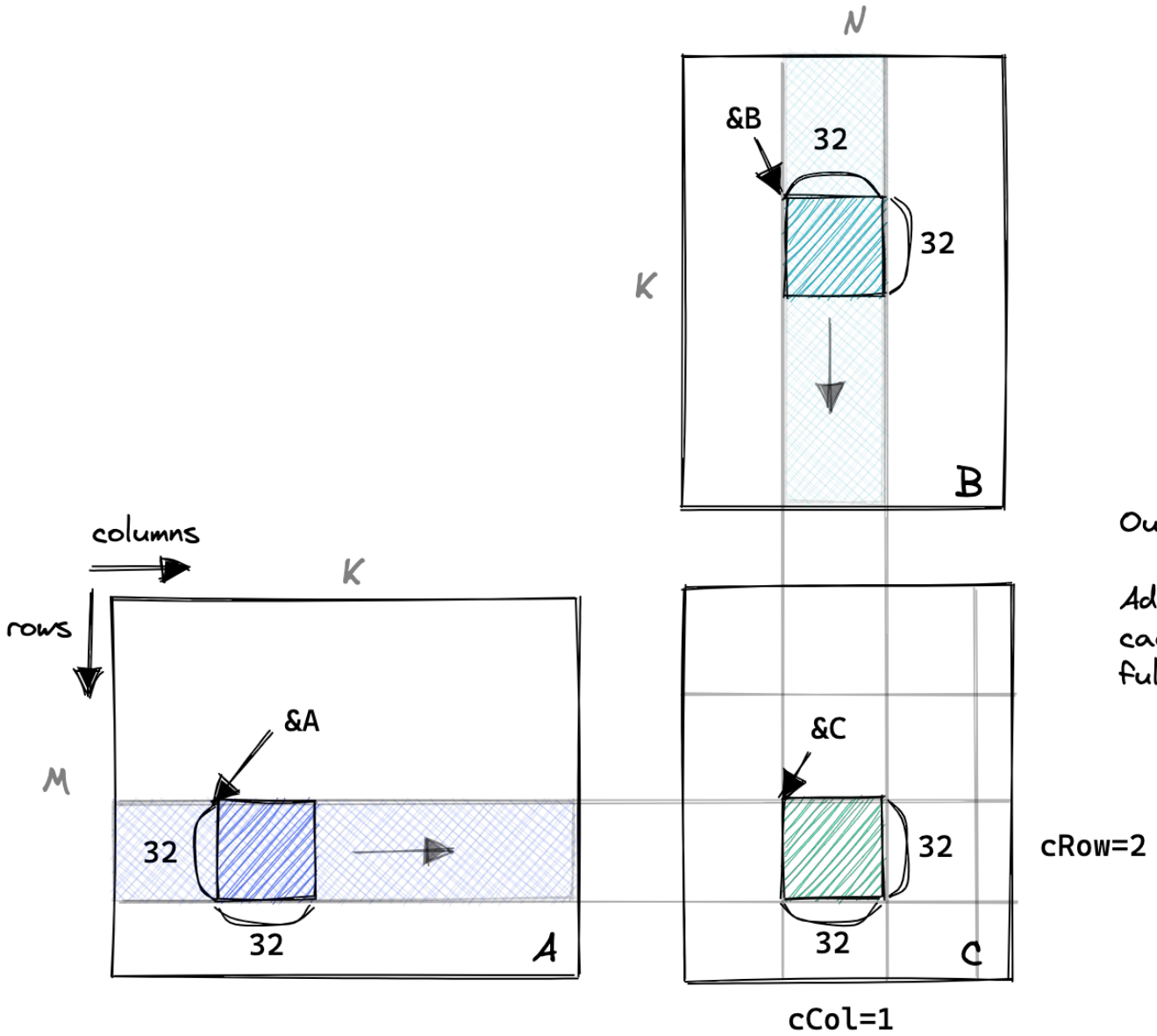
Make sure these threads end up in same warp to exploit coalescing

Matmul v0

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg
matmul_naive_kernel(float const*, float const*, floa...	0.00%	0.000us	0.00%	0.000us	0.000us	1.480ms	82.40%	1.480ms	740.228us
ampere_sgemm_128x64_nn	0.00%	0.000us	0.00%	0.000us	0.000us	313.440us	17.45%	313.440us	156.720us
Memset (Device)	0.00%	0.000us	0.00%	0.000us	0.000us	2.816us	0.16%	2.816us	1.408us
cudaLaunchKernel	4.29%	60.915us	4.29%	60.915us	15.229us	0.000us	0.00%	0.000us	0.000us
cudaMemsetAsync	1.35%	19.184us	1.35%	19.184us	9.592us	0.000us	0.00%	0.000us	0.000us
cudaOccupancyMaxActiveBlocksPerMultiprocessor	0.70%	10.011us	0.70%	10.011us	5.005us	0.000us	0.00%	0.000us	0.000us
cudaDeviceSynchronize	93.66%	1.331ms	93.66%	1.331ms	1.331ms	0.000us	0.00%	0.000us	0.000us

Matmul v1

1024*K*2

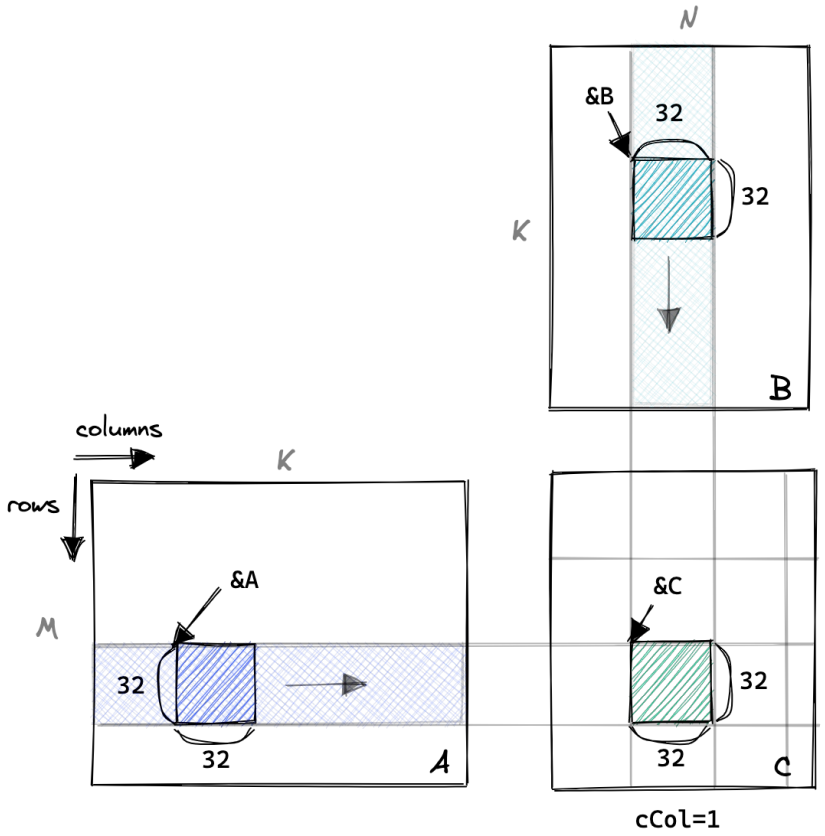


32*K*2
⚡1024*K*2⚡

Outer loop:

Advance &A,&B by size of cacheblock (=32*32) until C is fully calculated

Matmul v1



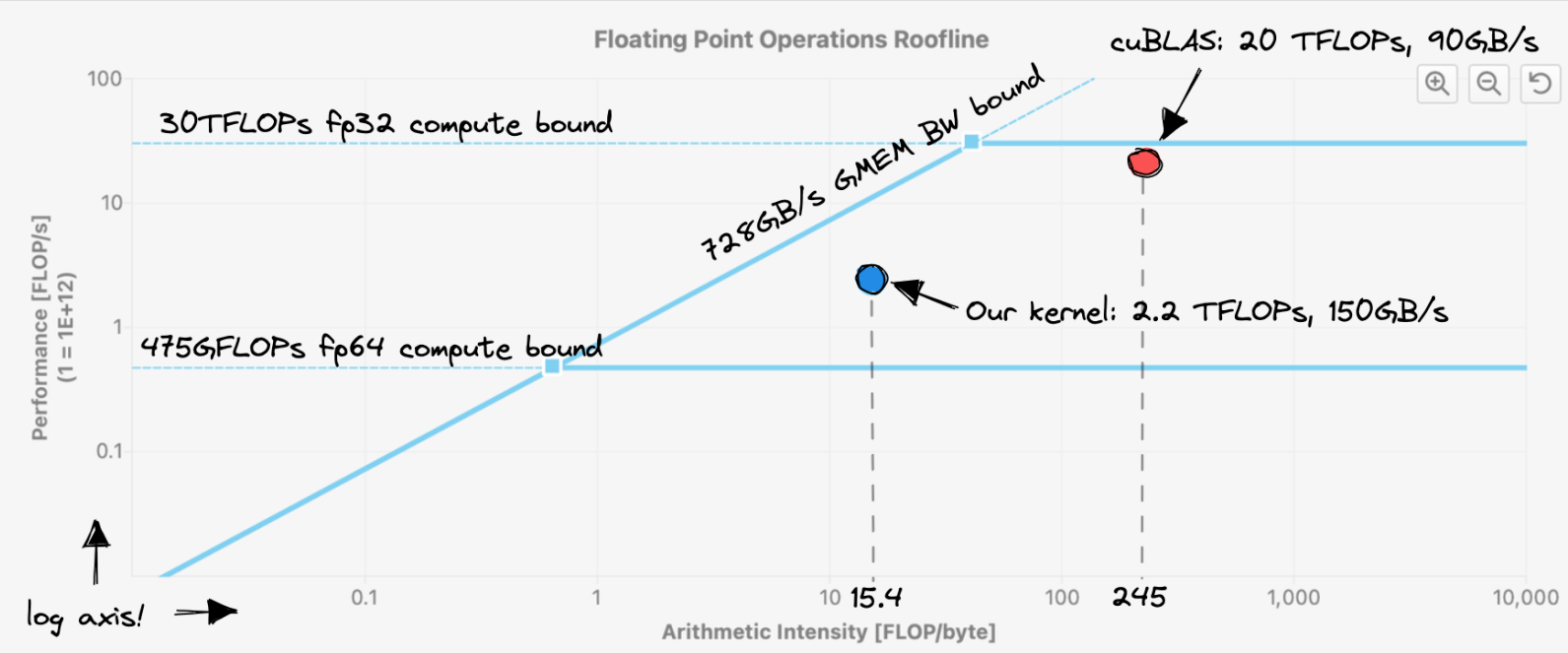
1_smem_matmul.py

```
cuda_source = """
__global__ void matmul_shared_kernel(const float* A, const float* B, float* C, int M, int N, int K) {
    __shared__ float As[32][32];
    __shared__ float Bs[32][32];

    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    float acc = 0.0f;

    for (int t = 0; t < (K + blockDim.x - 1) / blockDim.x; ++t) {
        if (row < M && t * blockDim.x + threadIdx.x < K)
            As[threadIdx.y][threadIdx.x] = A[row * K + t * blockDim.x + threadIdx.x];
        else
            As[threadIdx.y][threadIdx.x] = 0.0f;

        if (col < N && t * blockDim.x + threadIdx.y < K)
            Bs[threadIdx.y][threadIdx.x] = B[(t * blockDim.x + threadIdx.y) * N + col];
        else
            Bs[threadIdx.y][threadIdx.x] = 0.0f;

        __syncthreads();

        for (int k = 0; k < blockDim.x; ++k)
            acc += As[threadIdx.y][k] * Bs[k][threadIdx.x];

        __syncthreads();
    }

    if (row < M && col < N)
        C[row * N + col] = acc;
}

torch::Tensor matmul_shared(torch::Tensor A, torch::Tensor B) {
    TORCH_CHECK(A.size(1) == B.size(0), "Incompatible matrix sizes");
    int M = A.size(0), K = A.size(1), N = B.size(1);

    auto C = torch::empty({M, N}, A.options());

    dim3 threads(32, 32);
    dim3 blocks((N + 32 - 1) / 32, (M + 32 - 1) / 32);

    matmul_shared_kernel<<<blocks, threads>>>(
        A.data_ptr<float>(), B.data_ptr<float>(), C.data_ptr<float>(), M, N, K
    );

    return C;
}
"""
```

Matmul v1

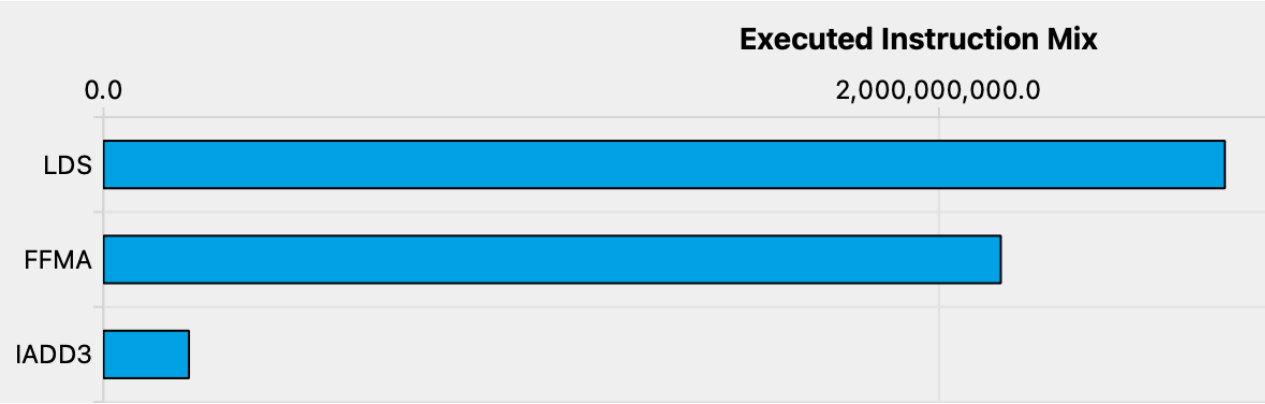
Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg	# of Calls
matmul_shared_kernel(float const*, float const*, flo...	0.00%	0.000us	0.00%	0.000us	0.000us	1.128ms	78.12%	1.128ms	563.923us	2
ampere_sgemm_128x64_nn	0.00%	0.000us	0.00%	0.000us	0.000us	313.025us	21.68%	313.025us	156.512us	2
Memset (Device)	0.00%	0.000us	0.00%	0.000us	0.000us	2.816us	0.20%	2.816us	1.408us	2
cudaLaunchKernel	6.03%	63.850us	6.03%	63.850us	15.962us	0.000us	0.00%	0.000us	0.000us	4
cudaMemsetAsync	1.80%	19.038us	1.80%	19.038us	9.519us	0.000us	0.00%	0.000us	0.000us	2
cudaOccupancyMaxActiveBlocksPerMultiprocessor	0.78%	8.212us	0.78%	8.212us	4.106us	0.000us	0.00%	0.000us	0.000us	2
cudaDeviceSynchronize	91.40%	968.317us	91.40%	968.317us	968.317us	0.000us	0.00%	0.000us	0.000us	1

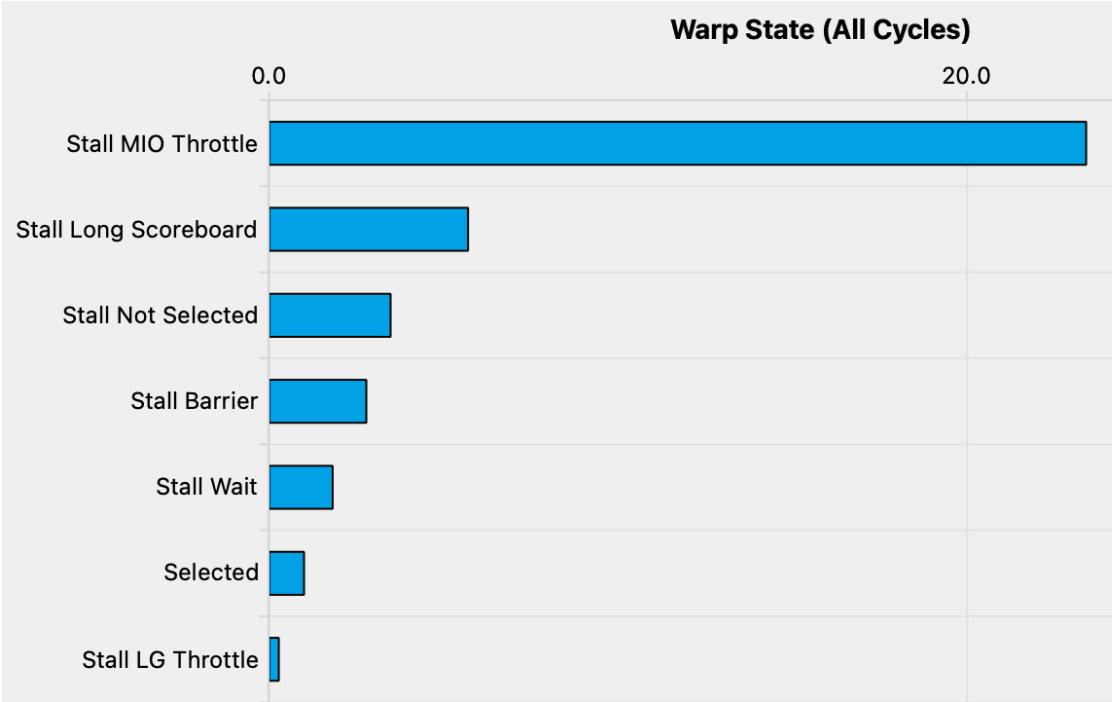
Matmul v2


Floating Point Operations Roofline
cuBLAS: 20 TFLOPs, 90GB/s
30TFLOPs fp32 compute bound
728GB/s GMEM BW bound
Our kernel: 2.2 TFLOPs, 150GB/s
475GFLOPs fp64 compute bound
Performance [FLOP/s]
(1 = 1E+12)
100
10
1
0.1
log axis!
0.1
1
10
15.4
100
245
1,000
10,000
Arithmetic Intensity [FLOP/byte]

Matmul v2

NVIDIA Nsight Compute

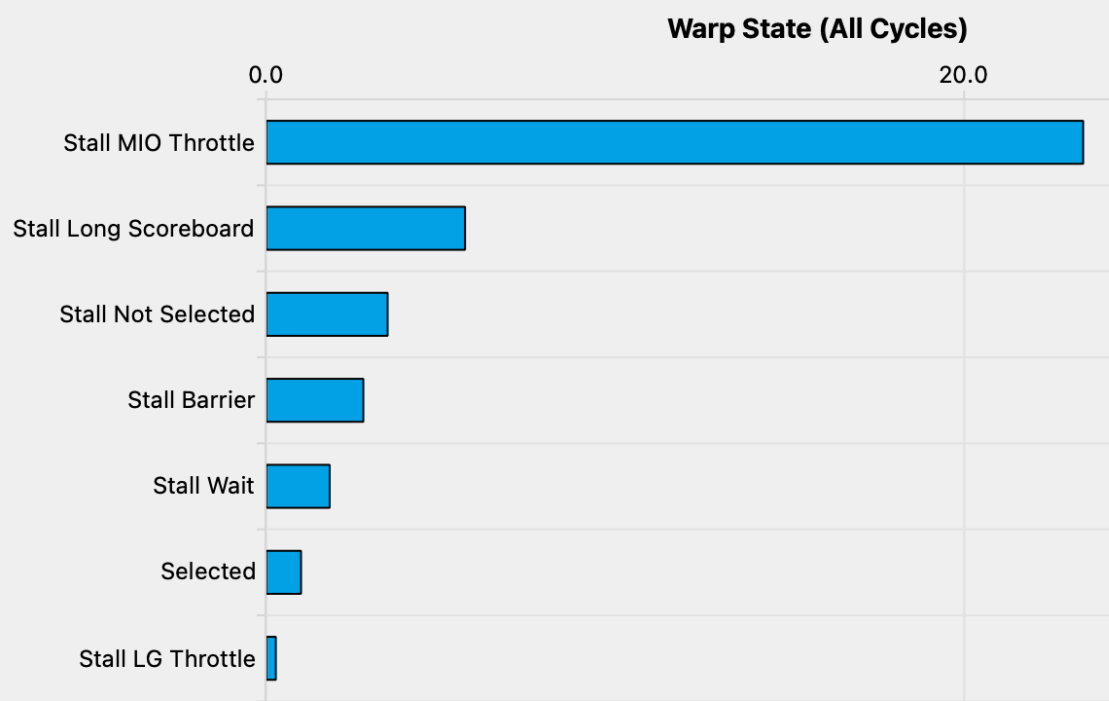




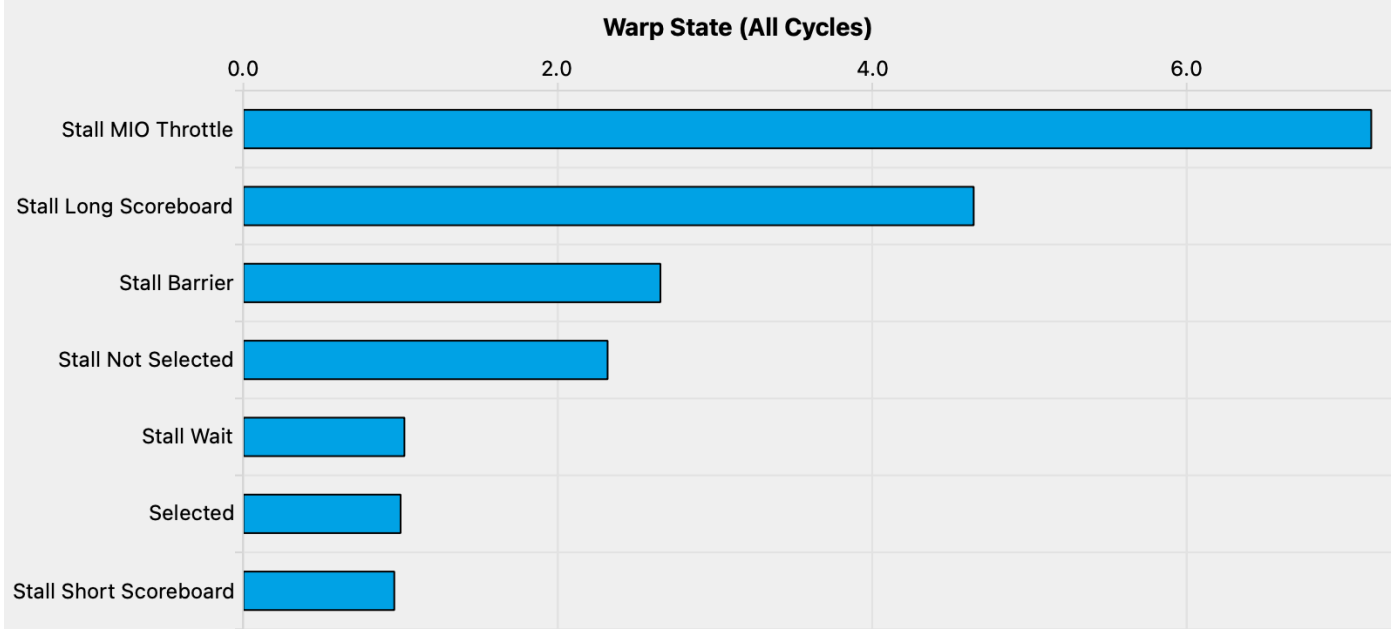
Stall Pipe Busy
Stall Long Scoreboard

Matmul v2

Thread任务：单线程计算单个C元素



Thread任务：单线程计算多个C元素

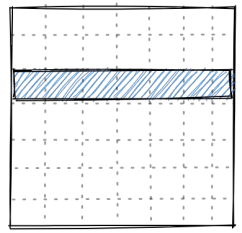


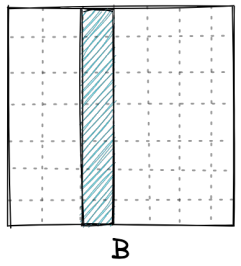
Matmul v2

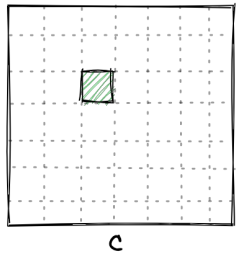
Calculating 1 result per thread requires:

- 7 loads from A
- 7 loads from B
- 1 load & 1 store to C

⇒ 15 loads & 1 store per result



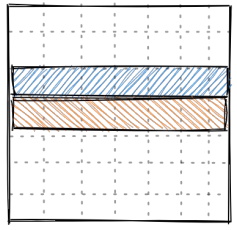


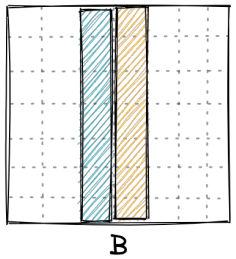


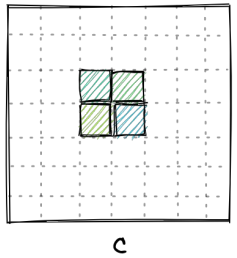
Calculating 4 results per thread requires:

- 14 loads from A
- 14 loads from B
- 4 loads & 4 stores to C

⇒ 8 loads & 1 store per result



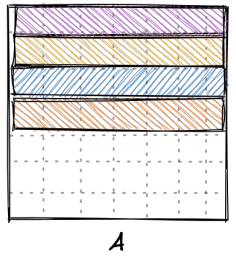


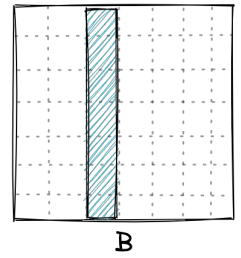


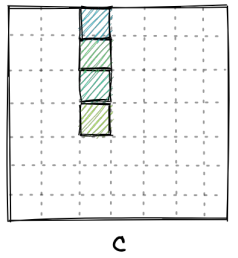
Calculating 4 results per thread requires:

- 28 loads from A
- 7 loads from B
- 4 loads & 4 stores to C

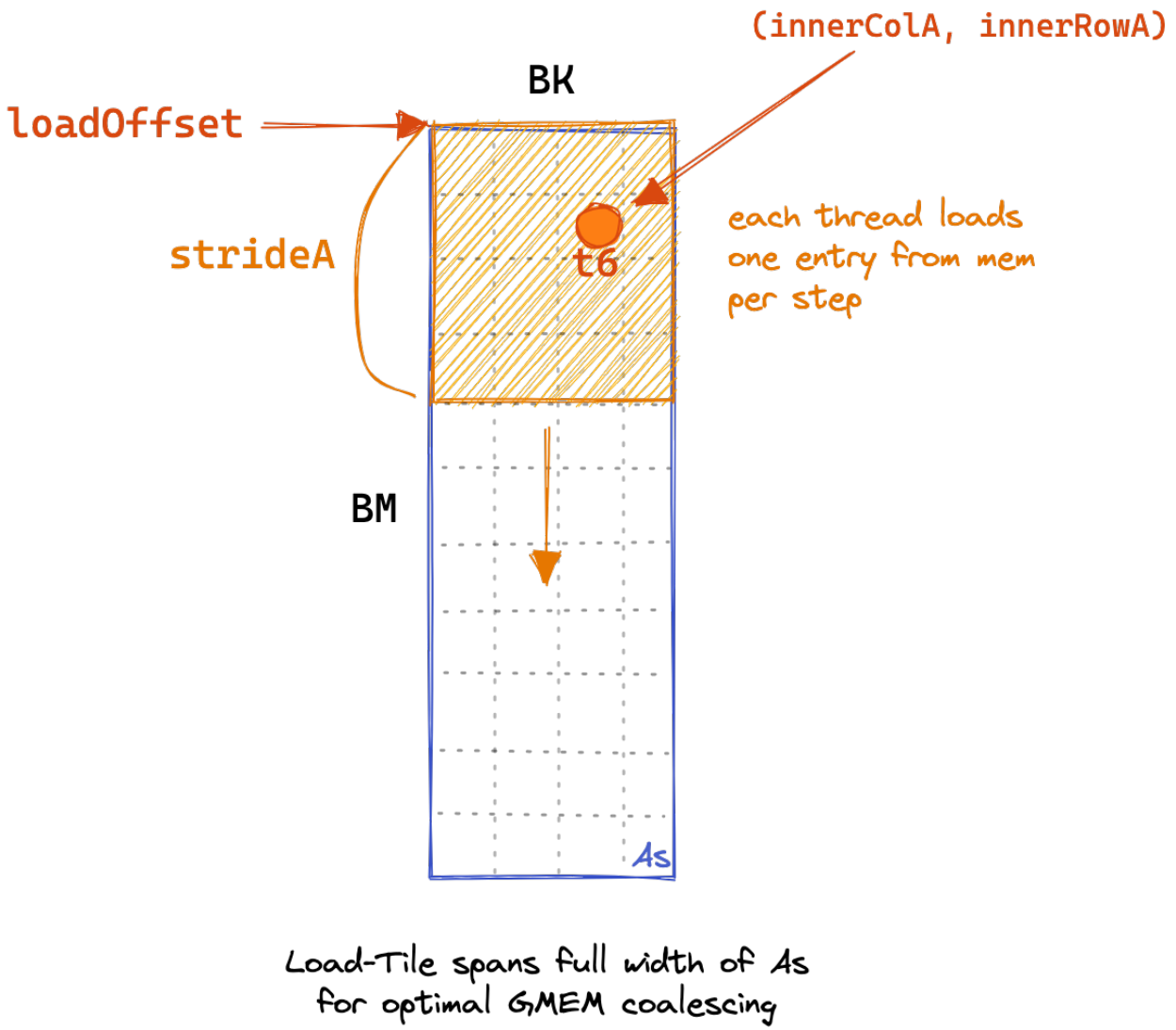
⇒ 9.75 loads & 1 store per result



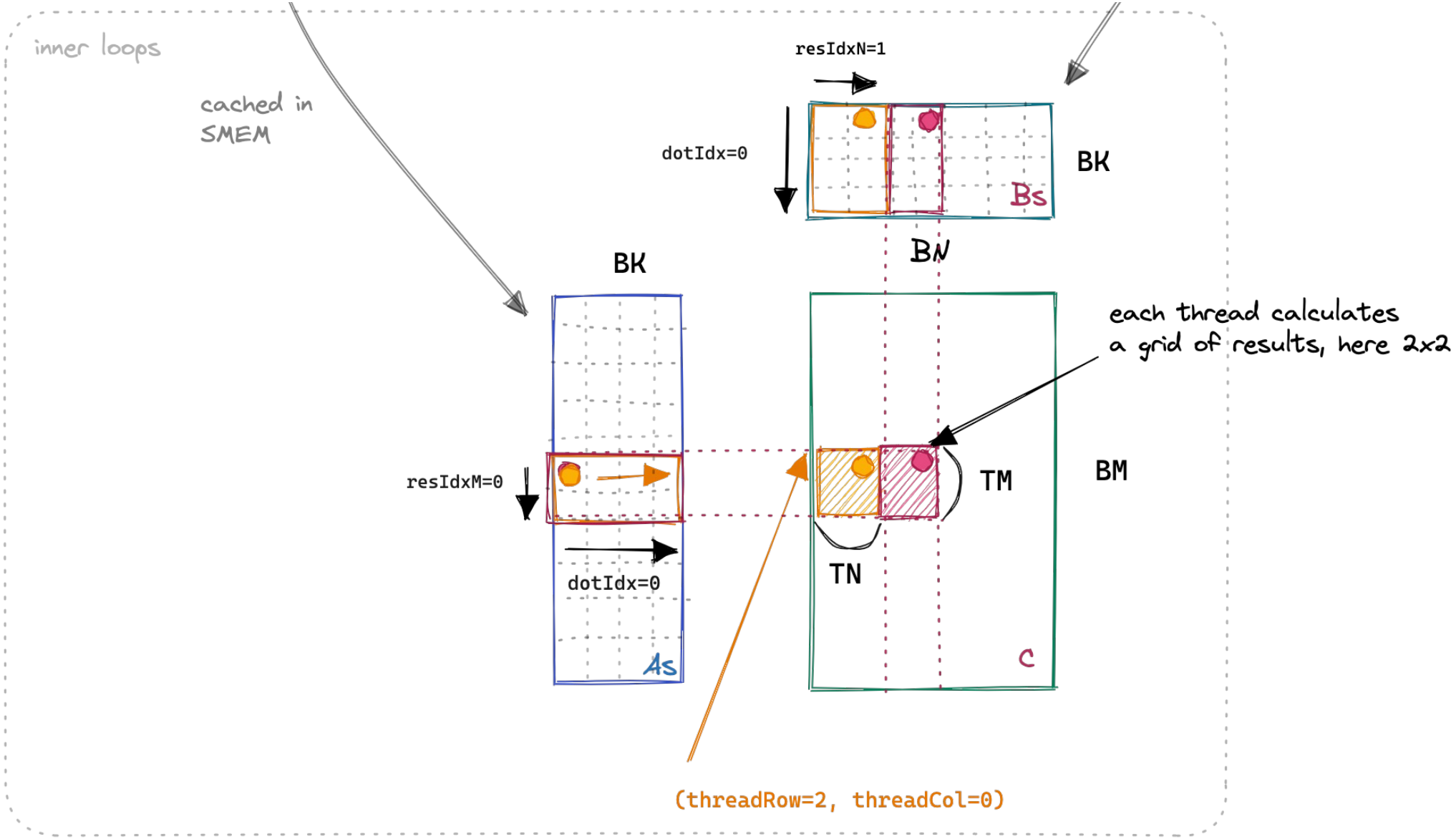




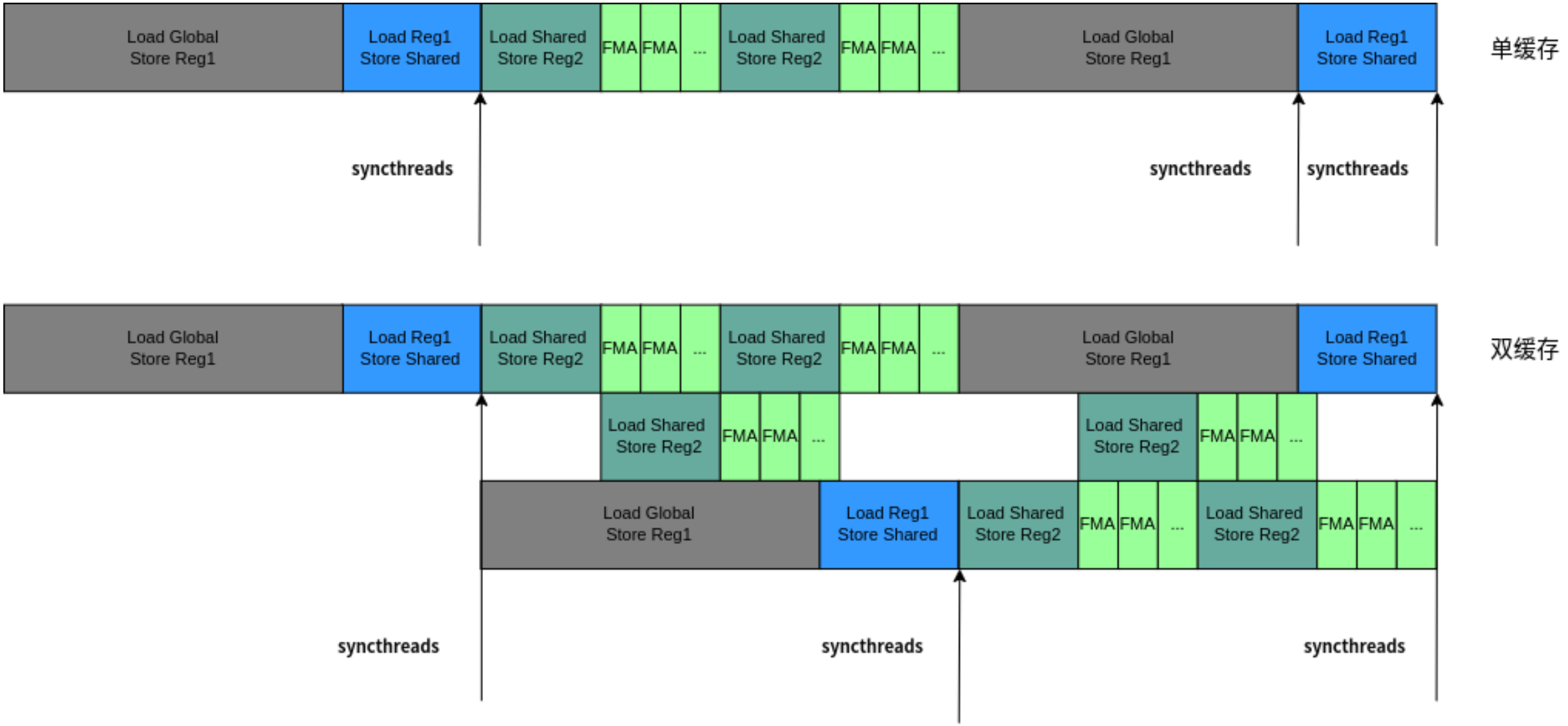
Matmul v2


(innerColA, innerRowA)
BK
loadOffset
strideA
t6
each thread loads
one entry from mem
per step
BM
As
Load-Tile spans full width of As
for optimal GMEM coalescing

Matmul v2


inner loops
cached in SMEM
resIdxN=1
dotIdx=0
BK
Bs
BN
BK
each thread calculates a grid of results, here 2x2
resIdxM=0
TM
BM
TN
dotIdx=0
As
C
(threadRow=2, threadCol=0)

Matmul v3


Load Global Store Reg1
Load Reg1 Store Shared
Load Shared Store Reg2
FMA
FMA
...
Load Shared Store Reg2
FMA
FMA
...
Load Global Store Reg1
Load Reg1 Store Shared
syncthreads
syncthreads
syncthreads
单缓存
Load Global Store Reg1
Load Reg1 Store Shared
Load Shared Store Reg2
FMA
FMA
...
Load Shared Store Reg2
FMA
FMA
...
Load Global Store Reg1
Load Reg1 Store Shared
Load Shared Store Reg2
FMA
FMA
...
Load Shared Store Reg2
FMA
FMA
...
Load Global Store Reg1
Load Reg1 Store Shared
Load Shared Store Reg2
FMA
FMA
...
Load Shared Store Reg2
FMA
FMA
...
syncthreads
syncthreads
syncthreads
双缓存

Matmul v3

GPU Features	NVIDIA Tesla P100	NVIDIA Tesla V100	NVIDIA A100
GPU Codename	GP100	GV100	GA100
GPU Architecture	NVIDIA Pascal	NVIDIA Volta	NVIDIA Ampere
Compute Capability	6.0	7.0	8.0
Threads / Warp	32	32	32
Max Warps / SM	64	64	64
Max Threads / SM	2048	2048	2048
Max Thread Blocks / SM	32	32	32
Max 32-bit Registers / SM	65536	65536	65536
Max Registers / Block	65536	65536	65536
Max Registers / Thread	255	255	255
Max Thread Block Size	1024	1024	1024
FP32 Cores / SM	64	64	64
Ratio of SM Registers to FP32 Cores	1024	1024	1024
Shared Memory Size / SM	64 KB	Configurable up to 96 KB	Configurable up to 164 KB

2048//64=32

65536//(72*64) = 14

164K//16K = 10

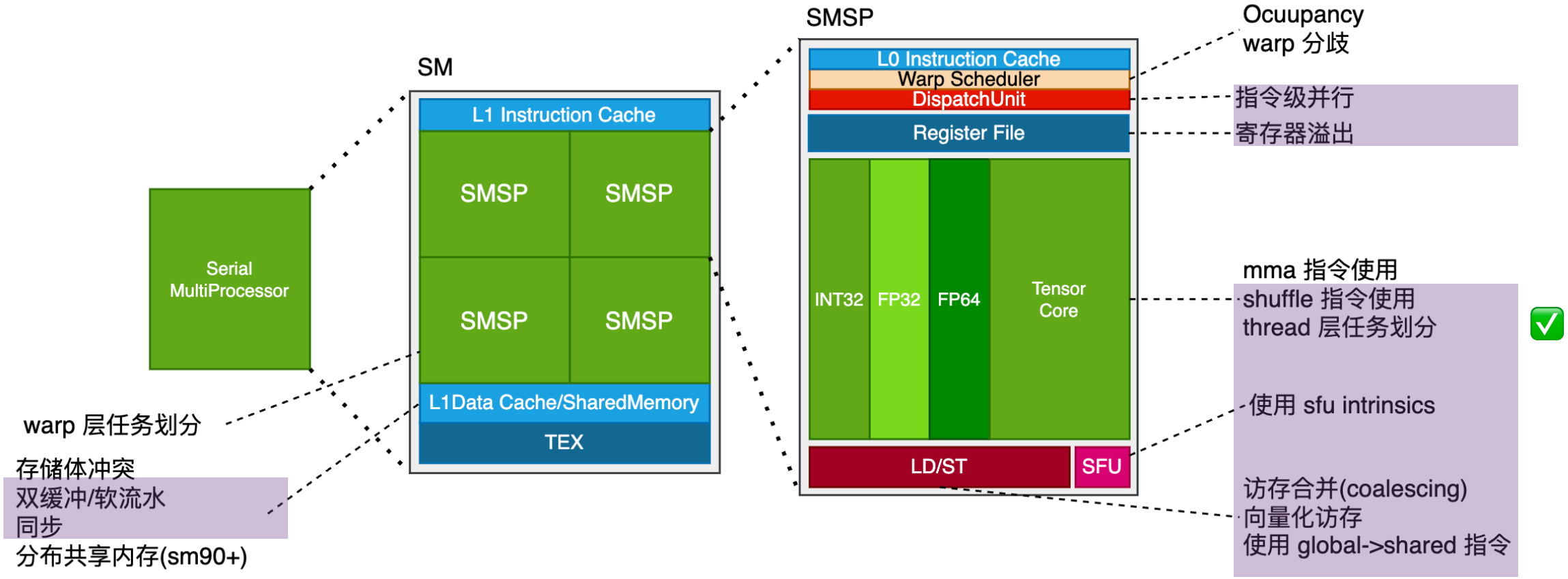
Matmul v4

类型	对应位宽
`float2`	64-bit
`float4`	128-bit
`int2`	64-bit
`int4`	128-bit
`half2`	32-bit
`__half2`	32-bit
`bfloat162`	32-bit
`bfloat168` *	128-bit
`char4`	32-bit
`uchar4`	32-bit
`ushort2`	32-bit
自定义结构体	任意

Matmul

作业三：试试看大模型能生成什么速度的CUDA Matmul

Conclusion


SMSP
Ocuupancy
warp 分歧
SM
L0 Instruction Cache
Warp Scheduler
DispatchUnit
指令级并行
Register File
寄存器溢出
L1 Instruction Cache
SMSP
SMSP
SMSP
SMSP
Serial
MultiProcessor
mma 指令使用
INT32
FP32
FP64
Tensor
Core
shuffle 指令使用
thread 层任务划分
L1Data Cache/SharedMemory
TEX
warp 层任务划分
使用 sfu intrinsics
LD/ST
SFU
存储体冲突
双缓冲/软流水
同步
访存合并(coalescing)
向量化访存
分布共享内存(sm90+)
使用 global->shared 指令

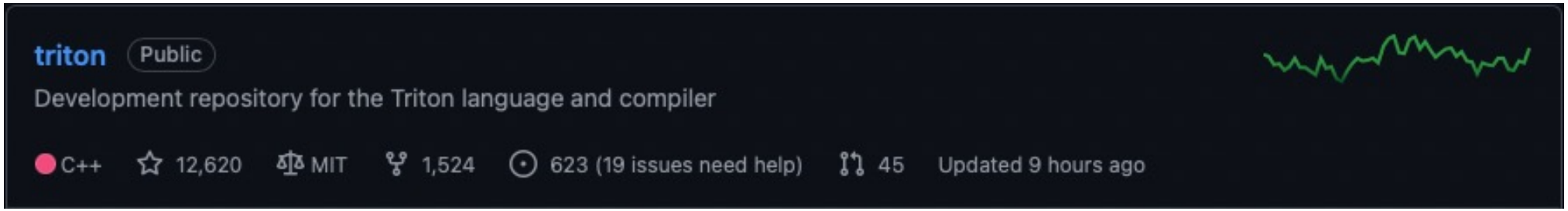
Triton

Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations

Philippe Tillet
Harvard University
USA

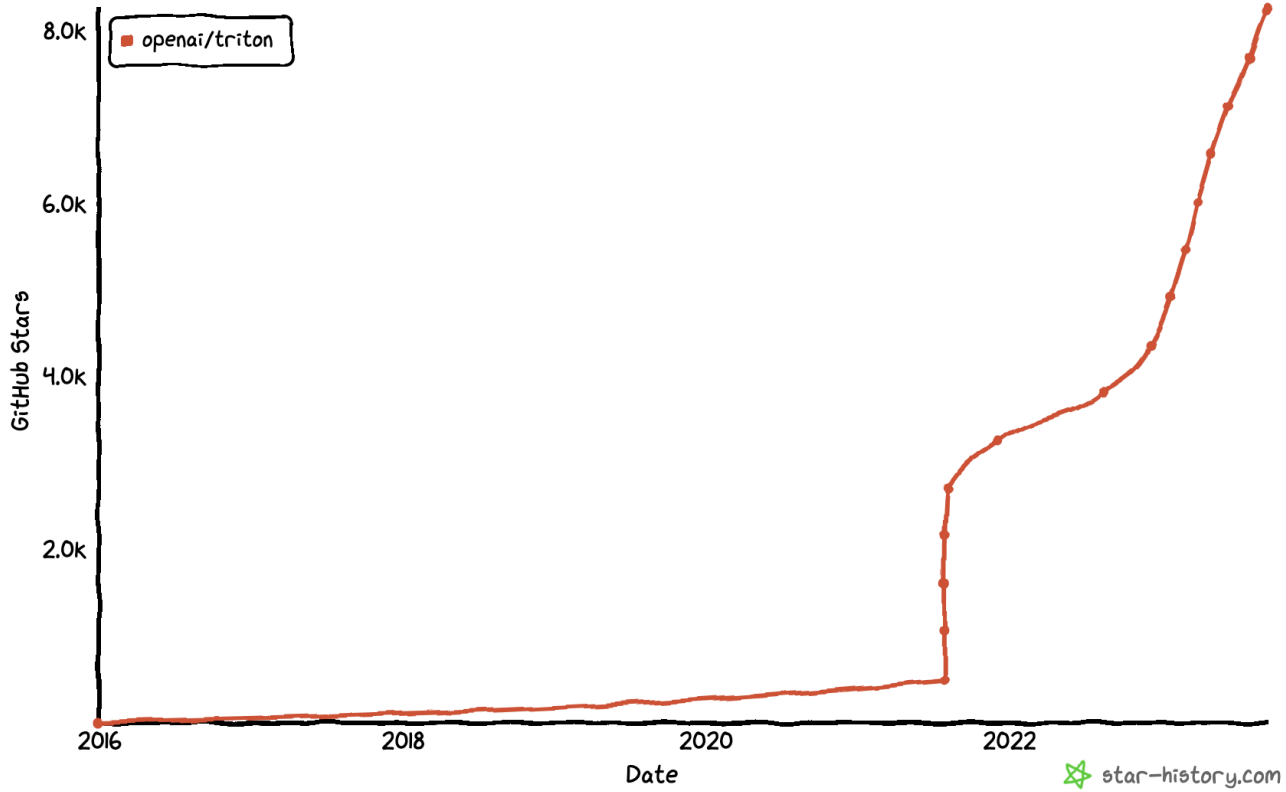
H. T. Kung
Harvard University
USA

David Cox
Harvard University, IBM
USA



Triton

- Triton是一个用于编写高效自定义深度学习算子的语言和编译器。相比CUDA，Triton生产力更高、更灵活。
- 2022年开始，Triton的热度呈爆发式增长，Triton生态逐渐形成规模。



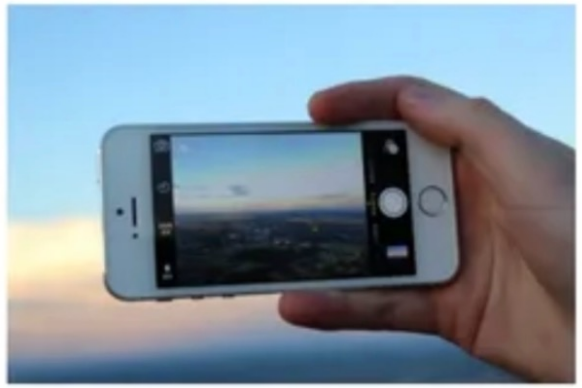
Triton

- **Triton 是一门 kernel 编程语言，主要用于编写 GPU kernels，也可以作为 AI 编译器的 Kernel 代码生成目标语言。和 CUDA, OpenCL 等属于并列的关系。**

Model	Pytorch/Tensorflow/JAX
Graph	XLA/HLO TVM/Relay Pytorch/fx
Kernel	CUDA OpenCL ROCM HIP SYCL **Triton**
Device	GPUs/CPUs/…

Triton vs. CUDA

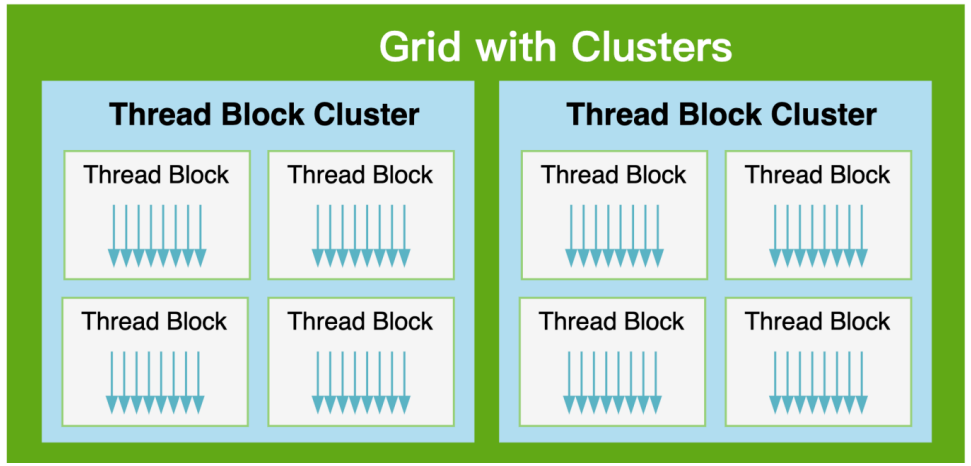
CUDA



Triton

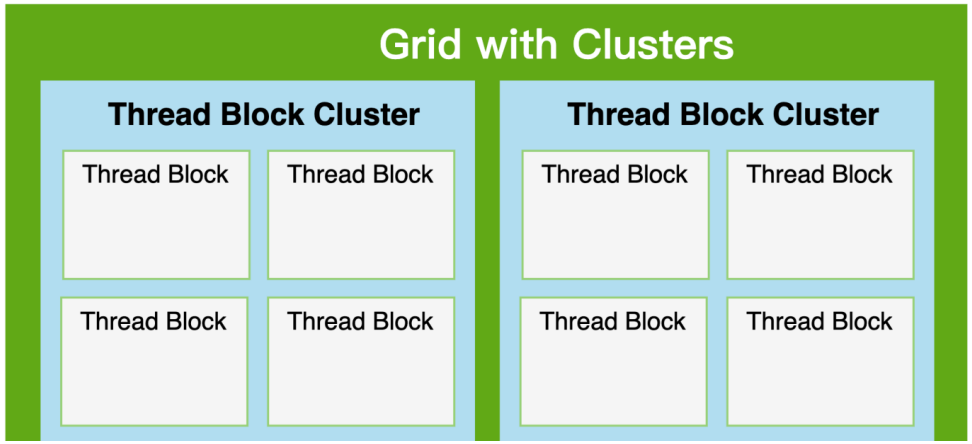
Triton vs. CUDA

► Thread-level Programming, Single instruction, multiple threads (SIMT)



software concept	hardware concept
grid(Kernel)	device
thread block cluster (TBC)	GPU Processing Cluster (GPC)
thread block (CTA)	Serial MultiProcessor(SM)
warp	Serial MultiProcessor Sub-Partitiom (SMSP)
thread	cuda core

► Block-level Programming, Single instruction, multiple data (SIMD)



software concept	hardware concept
Kernel	device
cluster	GPU Processing Cluster (GPC)
program(CTA)	Serial MultiProcessor(SM)

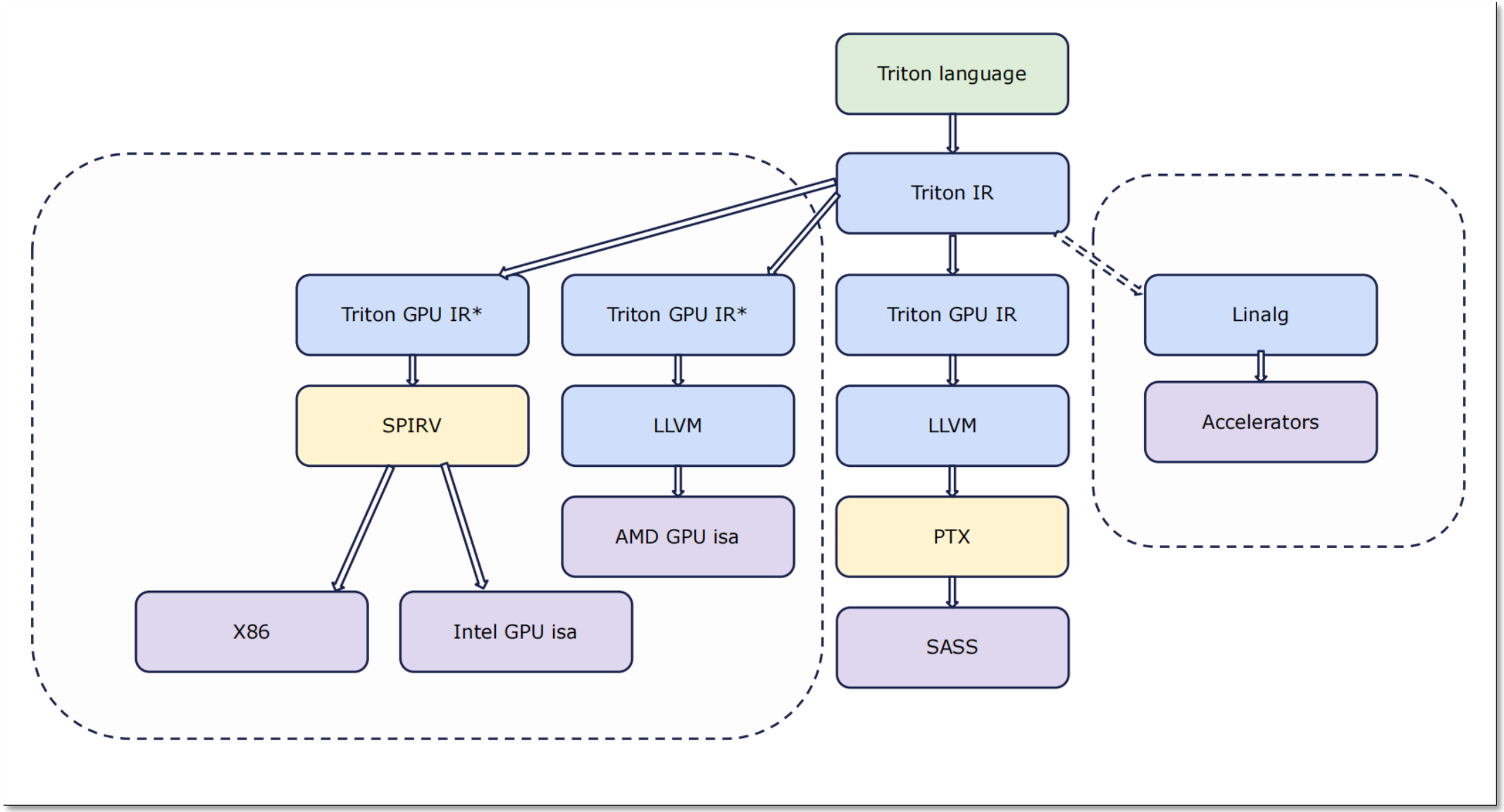
Triton vs. CUDA

	CUDA	Triton	Torch Op
Algorithm	User	User	Compiler
Shared Memory	User	Compiler	Compiler
Barriers	User	Compiler	Compiler
Distribution to blocks	User	User	Compiler
Grid size	User	User	Compiler
Distribution to Warps/threads	User	Compiler	Compiler
Tensor Core usage	User	Compiler	Compiler
Coalescing	User	Compiler	Compiler
Intermediate data layout	User	Compiler	Compiler
Workgroup size	User	User	Compiler

Why Triton?

- 更高抽象层级的编程模型，更低的心智负担。
  - Block 级别编程，SPMD 编程模型，更简单的任务划分;
  - 定义在 SRAM (寄存器和 shared memory)上的 tensor;
  - 用于 tensor 的 primitive.(reduce, scan, sort, dot, ...);
  - triton compiler 的优化使非 GPU/CUDA 编程专家也能写出性能不俗的 kernel.
- Triton 编译器代码开源，对多元 AI 芯片有更好的可移植性。

Why Triton?


Triton language
Triton IR
Triton GPU IR*
Triton GPU IR*
Triton GPU IR
Linalg
SPIRV
LLVM
LLVM
Accelerators
AMD GPU isa
PTX
X86
Intel GPU isa
SASS

Triton vs. CUDA

block_id	thread_id	offs	offs < n	x_value	y_value	z_value
0	0	0	True	1	0	1
0	1	1	True	2	1	3
0	2	2	True	3	0	3
0	3	3	True	4	1	5
1	0	4	True	5	0	5
1	1	5	True	6	1	7
1	2	6	False	undefined	undefined	undefined
1	3	7	False	undefined	undefined	undefined

block_id	offs	offs < n	x_value	y_value	z_value
0	[0, 1, 2, 3]	[True, True, True, True]	[1, 2, 3, 4]	[0, 1, 0, 1]	[1, 3, 3, 5]
1	[4, 5, 6, 7]	[True, True, False, False]	[5, 6, 'undefined', 'undefined']	[0, 1, 'undefined', 'undefined']	[5, 7, 'undefined', 'undefined']

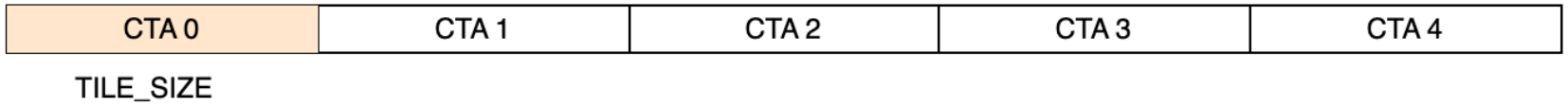
Triton

Triton Index

index	name	tags	link
1	Triton RMSNorm + Residual	norm	link
2	Flash-Decoding / Split-KV Attention	attention, decoding	link
3	Cross entropy loss	loss	link
4	Geglu	Activation	link
5	RMSNorm	Normalization	link
6	Rope Embedding	Embedding	link
7	Swiglu	Activation	link
8	Gemm Int4	Quant, Dequant, Matmul, Gemm	link
9	Gemm Int8	Quant, Dequant, Matmul, Gemm	link
10	Batched Conv2D	Conv	link
11	Batched Addition	Add	link
12	Batched Patchng	Rearranging	link
13	Block-Sparse Matrix Multiplication	Sparsity, Matmul	link
14	Fused Linear Cross Entropy	Linear Cross Entropy	link
n+1	your contribution	its tags	

算子开发示例: Vector Add（1 tile/CTA）

1 tile/CTA




```
@triton.jit
def add_kernel(a_ptr, b_ptr, o_ptr, N, TILE_SIZE: tl.constexpr):

    pid = tl.program_id(0)
    offsets = pid * TILE_SIZE + tl.arange(0, TILE_SIZE)
    mask = offsets < N

    a = tl.load(a_ptr + offsets, mask=mask)
    b = tl.load(b_ptr + offsets, mask=mask)
    o = a + b

    tl.store(o_ptr + offsets, o, mask=mask)
```

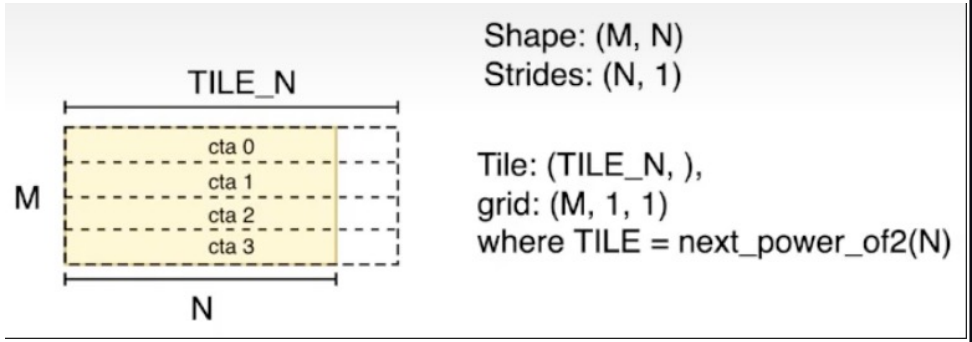
- Kernel: 描述每个CTA的运算
- 通过program_id区分CTA
- 通过range生成tensor

```
def add(a, b):
    o = torch.empty_like(a)
    N = a.numel()

    TILE_SIZE = 128
    grid = (triton.cdiv(N, TILE_SIZE), 1, 1)
    add_kernel[grid](a, b, o, N, TILE_SIZE, num_warps=4)

    return o
```

- Wrapper: 指定grid， 启动kernel

算子开发示例: Softmax

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

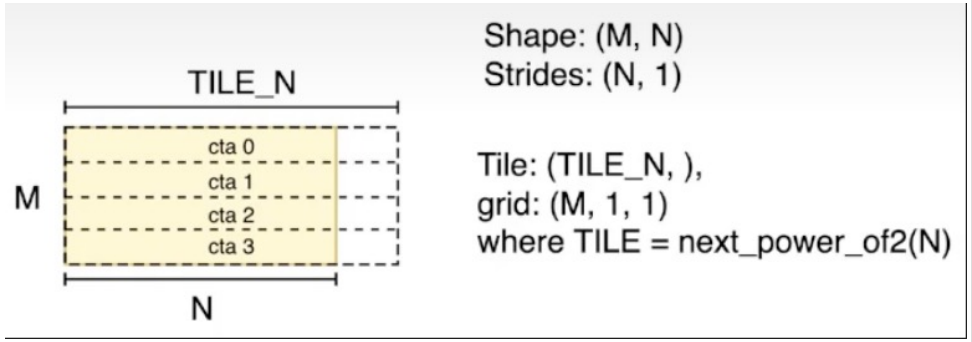


0_naive_softmax.py

```python
x = torch.randn((4096, 32768), device="cuda")
# x = torch.randn((4096, 8192), device="cuda")
# x = torch.randn((4096, 4096), device="cuda")

with profile(
    activities=[ProfilerActivity.CUDA],
    schedule=torch.profiler.schedule(wait=1, warmup=1, active=3),
    on_trace_ready=torch.profiler.tensorboard_trace_handler("./log/naive_softmax")
,   record_shapes=False,
    with_stack=False,
) as prof:
    for i in range(5):
        with torch.no_grad():
            out1 = softmax(x)
        with torch.no_grad():
            out0 = torch.softmax(x, dim=-1)
        prof.step()

out1 = softmax(x)
out0 = torch.softmax(x, dim=-1)
assert torch.allclose(out1, out0, atol=1e-3)
print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

算子开发示例: Softmax

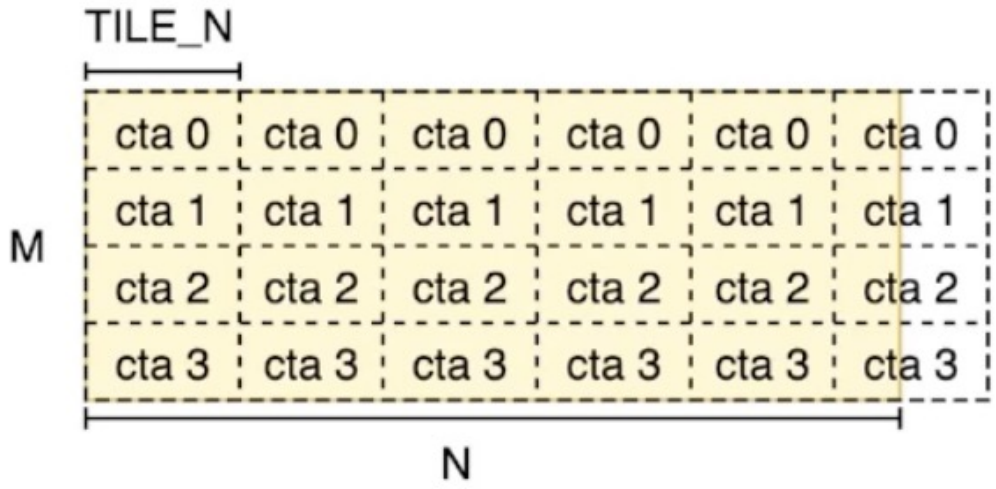
$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



0_naive_softmax.py

```python
@triton.jit
def softmax_kernel(output_ptr, input_ptr, M, N, TILE_N: tl.constexpr):
    pid_m = tl.program_id(0)  # (0...M-1, 0, 0)
    n_offsets = tl.arange(0, TILE_N)  # [0, ..., TILE_N-1]
    offset = pid_m * N + n_offsets  # (pid_m, [0, ..., TILE_N-1])

    x = tl.load(input_ptr + offset, mask=n_offsets < N, other=-float("inf"
))  m = tl.max(x)
    e = tl.exp(x - m)
    z = tl.sum(e)
    out = e / z
    tl.store(output_ptr + offset, out, mask=n_offsets < N)


def softmax(x):
    M, N = x.shape
    out = torch.empty_like(x)
    TILE_N = triton.next_power_of_2(N)
    grid = (M, 1, 1)
    softmax_kernel[grid](out, x, M, N, TILE_N)
    return out
```

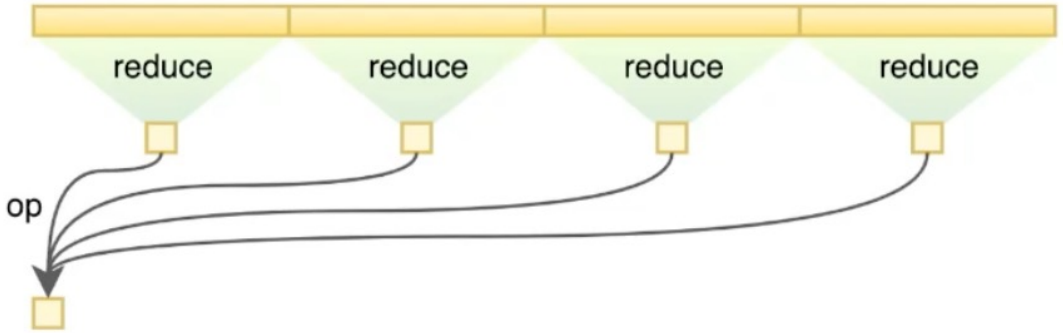
算子开发示例: Softmax



1_multi_cta_softmax.py

```
1  def softmax(x):
2      M, N = x.shape
3      out = torch.empty_like(x)
+ 4      TILE_N = min(4096, triton.next_power_of_2(N))
5      grid = (M, 1, 1)
6      softmax_kernel_loop_v1[grid](out, x, M, N, TILE_N
7  )   return out
```

算子开发示例: Softmax



1_multi_cta_softmax.py

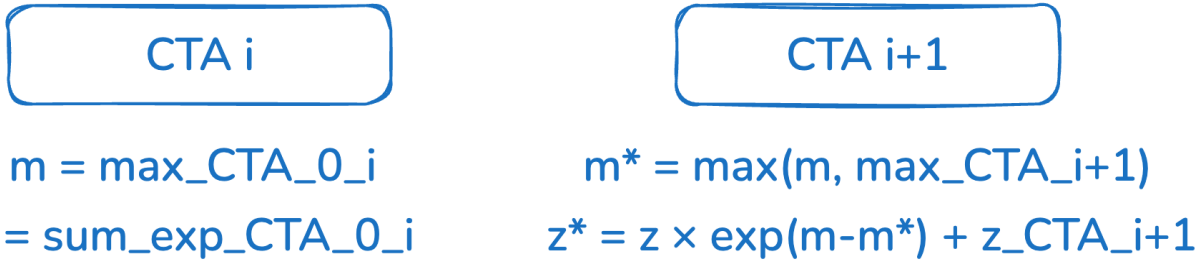
```
@triton.jit
def softmax_kernel_loop_v1(output_ptr, input_ptr, M, N, TILE_N: tl.constexpr):
    pid_m = tl.program_id(0)  # (0..M-1, 0, 0)

    m = tl.full((), value=-float("inf"), dtype=output_ptr.dtype.element_ty)
+   for start_cta in range(0, N, TILE_N):  # num of CTA
        offsets_cta = start_cta + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
        m = tl.maximum(m, tl.max(x))

    z = tl.full((), value=0, dtype=output_ptr.dtype.element_ty)
+   for start_cta in range(0, N, TILE_N):  # num of CTA
        offsets_cta = start_cta + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
        e = tl.exp(x - m)
        z += tl.sum(e)

+   for start_cta in range(0, N, TILE_N):  # num of CTA
        offsets_cta = start_cta + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
        e = tl.exp(x - m)
        out = e / z
        tl.store(output_ptr + offsets_x, out, mask=offsets_cta < N)

```

算子开发示例: Softmax

x: [3, 2, 5, 1]

m0=-inf	z0=0
m1=3	z1=e(3-5)
m2=3	z2=z1+e(2-5)
m3=5	z3=z2+e(5-5)
m4=5	z4=z3+e(1-5)

算子开发示例: Softmax



2_online_softmax.py

```
@triton.jit
def softmax_kernel_online_v1(output_ptr, input_ptr, M, N, TILE_N: tl.constexpr):
    pid_m = tl.program_id(0)
    m = tl.full((), value=-float("inf"), dtype=output_ptr.dtype.element_ty)
    z = tl.full((), value=0, dtype=output_ptr.dtype.element_ty)
    for start_cta in range(0, N, TILE_N):
        offsets_cta = start_cta + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
+       new_m = tl.maximum(m, tl.max(x))
+       new_z = tl.exp(m - new_m) * z + tl.sum(tl.exp(x - new_m))
        m = new_m
        z = new_z

    for start_cta in range(0, N, TILE_N):
        offsets_cta = start_cta + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
        e = tl.exp(x - m)
        out = e / z
        tl.store(output_ptr + offsets_x, out, mask=offsets_cta < N)
```

算子开发示例: Softmax

3_online_softmax_v2.py

```
+  1  @triton.jit
+  2  def prev_multiple_of(a, b):
+  3      return tl.cdiv(a, b) * b - b
   4
   5  @triton.jit
   6  def softmax_kernel_online_v1(output_ptr, input_ptr, M, N, TILE_N: tl.constexpr):
   7      pid_m = tl.program_id(0)
   8      m = tl.full((), value=-float("inf"), dtype=output_ptr.dtype.element_ty)
   9      z = tl.full((), value=0, dtype=output_ptr.dtype.element_ty)
  10      before_last = prev_multiple_of(N, TILE_N)
  11      for start_cta in range(0, before_last, TILE_N):
  12          offsets_cta = start_cta + tl.arange(0, TILE_N)
  13          offsets_x = pid_m * N + offsets_cta
+ 14          x = tl.load(input_ptr + offsets_x)
- 15          # x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
  16          new_m = tl.maximum(m, tl.max(x))
  17          new_z = tl.exp(m - new_m) * z + tl.sum(tl.exp(x - new_m))
  18          m = new_m
  19          z = new_z
  20
  21      for start_cta in range(before_last, N, TILE_N):
  22          offsets_cta = start_cta + tl.arange(0, TILE_N)
  23          offsets_x = pid_m * N + offsets_cta
  24          x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
  25          new_m = tl.maximum(m, tl.max(x))
  26          new_z = tl.exp(m - new_m) * z + tl.sum(tl.exp(x - new_m))
  27          m = new_m
  28          z = new_z
  29
  30      for start_cta in range(0, before_last, TILE_N):
  31          offsets_cta = start_cta + tl.arange(0, TILE_N)
  32          offsets_x = pid_m * N + offsets_cta
+ 33          x = tl.load(input_ptr + offsets_x)
  34          e = tl.exp(x - m)
  35          out = e / z
  36          tl.store(output_ptr + offsets_x, out)
  37
  38      for start_cta in range(before_last, N, TILE_N):
  39          offsets_cta = start_cta + tl.arange(0, TILE_N)
  40          offsets_x = pid_m * N + offsets_cta
  41          x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
  42          e = tl.exp(x - m)
  43          out = e / z
  44          tl.store(output_ptr + offsets_x, out, mask=offsets_cta < N)
```

算子开发示例: Softmax

4_online_softmax_v3.py

```
@triton.jit
def softmax_kernel_online_v1(output_ptr, input_ptr, M, N, TILE_N: tl.constexpr):
    pid_m = tl.program_id(0)
    m = tl.full((), value=-float("inf"), dtype=output_ptr.dtype.element_ty)
    z = tl.full((), value=0, dtype=output_ptr.dtype.element_ty)
    before_last = prev_multiple_of(N, TILE_N)
    for start_cta in range(0, before_last, TILE_N):
        offsets_cta = start_cta + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x)
        # x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
        new_m = tl.maximum(m, tl.max(x))
        new_z = tl.exp(m - new_m) * z + tl.sum(tl.exp(x - new_m))
        m = new_m
        z = new_z

    for start_cta in range(before_last, N, TILE_N):
        offsets_cta = start_cta + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
        new_m = tl.maximum(m, tl.max(x))
        new_z = tl.exp(m - new_m) * z + tl.sum(tl.exp(x - new_m))
        m = new_m
        z = new_z

+   for start_cta in range(0, TILE_N, TILE_N):
+       offsets_cta = before_last + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x, mask=offsets_cta < N, other=-float("inf"))
        e = tl.exp(x - m)
        out = e / z
        tl.store(output_ptr + offsets_x, out, mask=offsets_cta < N)

+   for start_cta in range(TILE_N, N, TILE_N):
+       offsets_cta = (before_last - start_cta) + tl.arange(0, TILE_N)
        offsets_x = pid_m * N + offsets_cta
        x = tl.load(input_ptr + offsets_x)
        e = tl.exp(x - m)
        out = e / z
        tl.store(output_ptr + offsets_x, out)
```

算子开发示例: Softmax

5_online_softmax_v4.py

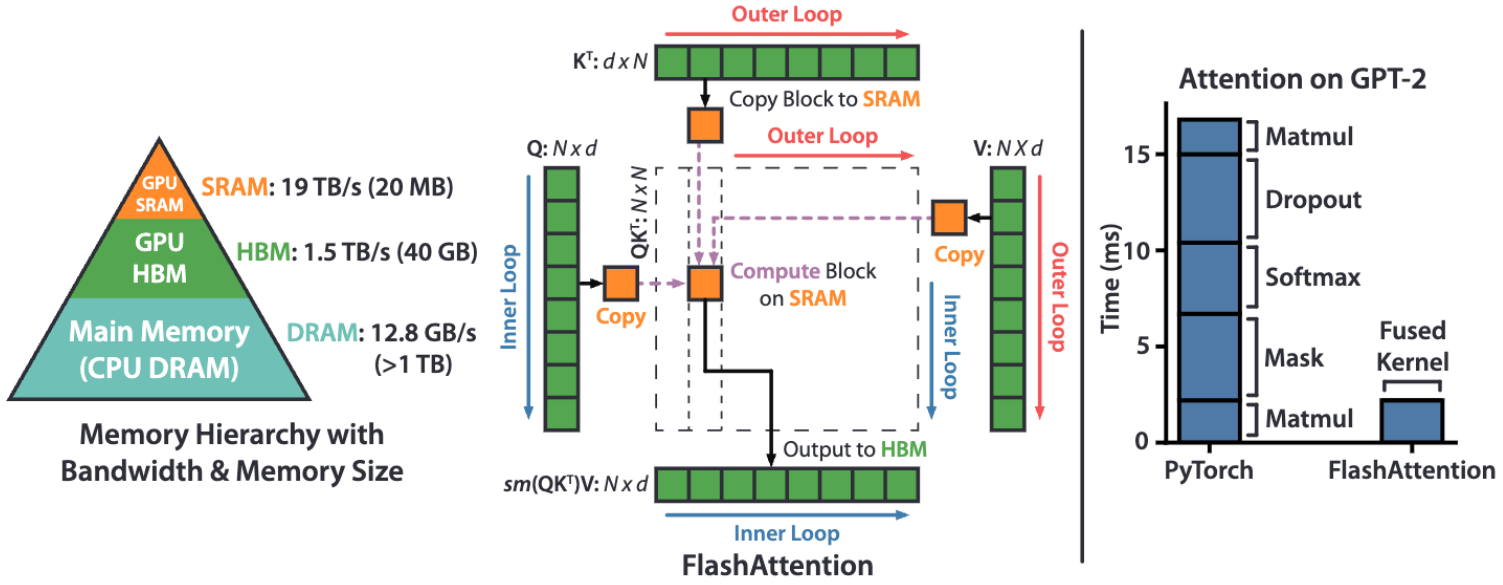
```
for start_cta in range(0, TILE_N, TILE_N):
    offsets_cta = before_last + tl.arange(0, TILE_N)
    offsets_x = pid_m * N + offsets_cta
    x = tl.load(
        input_ptr + offsets_x,
+       eviction_policy="evict_first",
        mask=offsets_cta < N,
        other=-float("inf"),
    )
    e = tl.exp(x - m)
    out = e / z
    tl.store(output_ptr + offsets_x, out, mask=offsets_cta < N)

for start_cta in range(TILE_N, N, TILE_N):
    offsets_cta = (before_last - start_cta) + tl.arange(0, TILE_N)
    offsets_x = pid_m * N + offsets_cta
+   x = tl.load(input_ptr + offsets_x, eviction_policy="evict_first"
)   e = tl.exp(x - m)
    out = e / z
    tl.store(output_ptr + offsets_x, out)
```

FlashAttention

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao[†], Daniel Y. Fu[†], Stefano Ermon[†], Atri Rudra[‡], and Christopher Ré[†]

[†]Department of Computer Science, Stanford University
[‡]Department of Computer Science and Engineering, University at Buffalo, SUNY



FlashAttention

Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min\left(\lceil \frac{M}{4d} \rceil, d\right)$.

Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.

Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ of size B_r each.

for $1 \le j \le T_c$ **do**

  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

  for $1 \le i \le T_r$ **do**

    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.

    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

    On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.

    Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1}(\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.

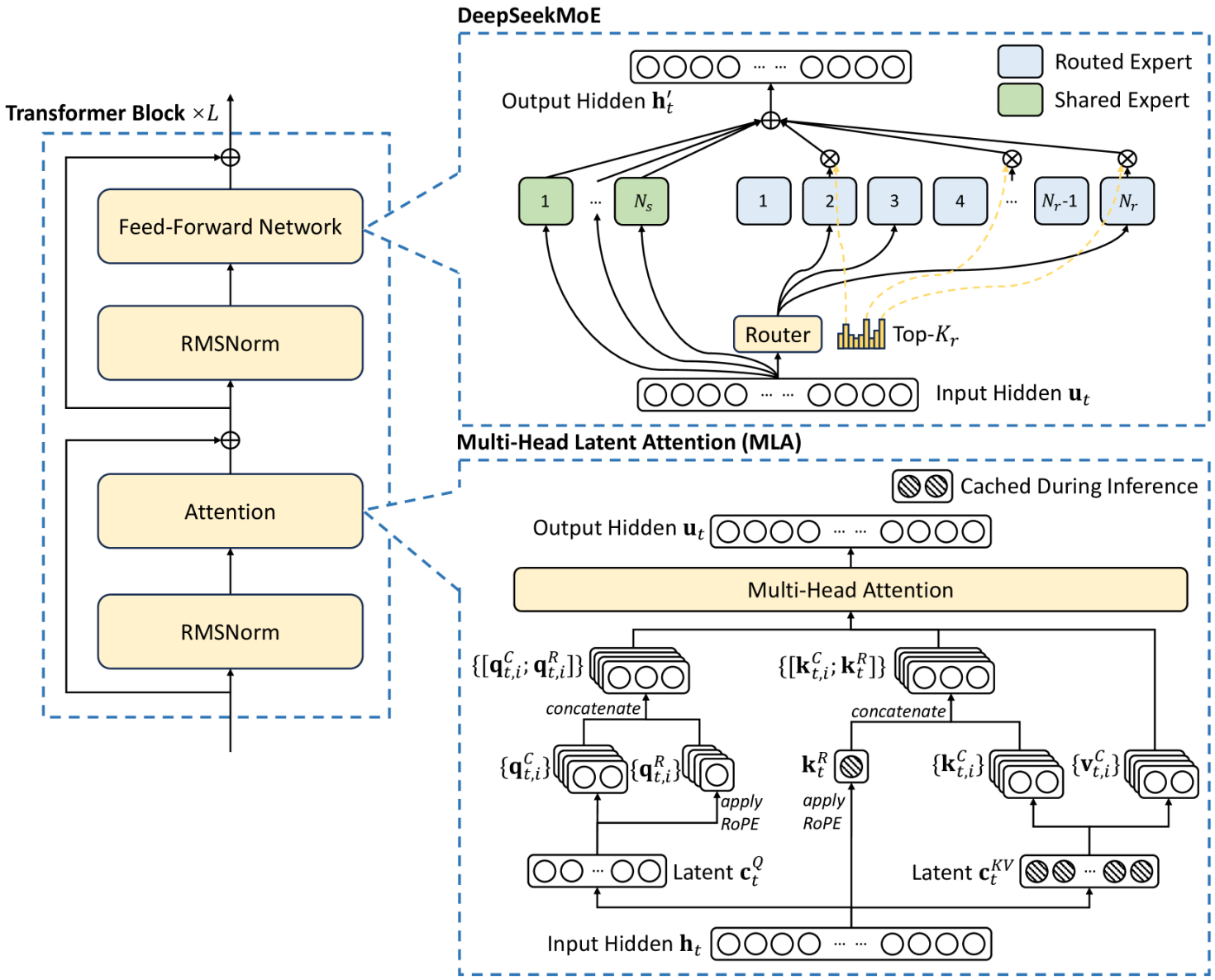
    Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.

  end for

end for

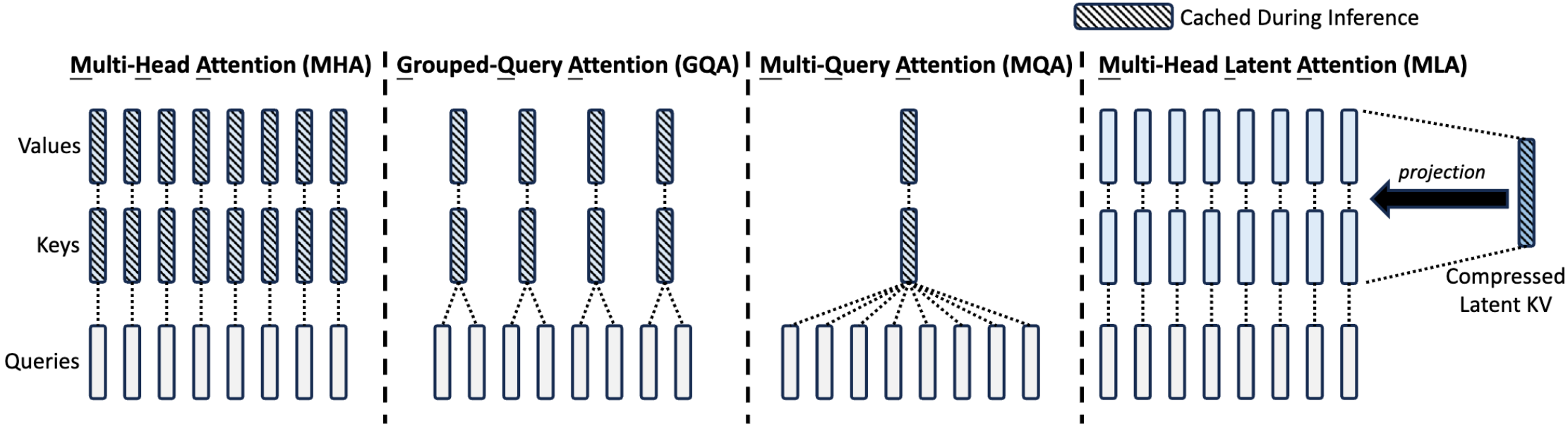
Return $\mathbf{O}$.

Multi-head Latent Attention



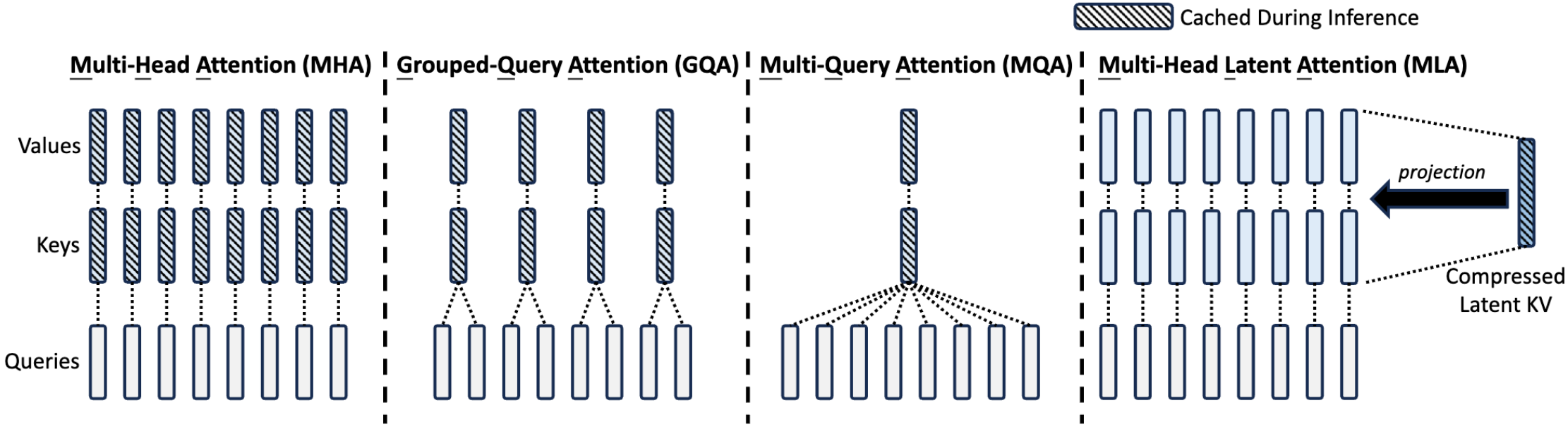
MLA

MHA vs. MLA



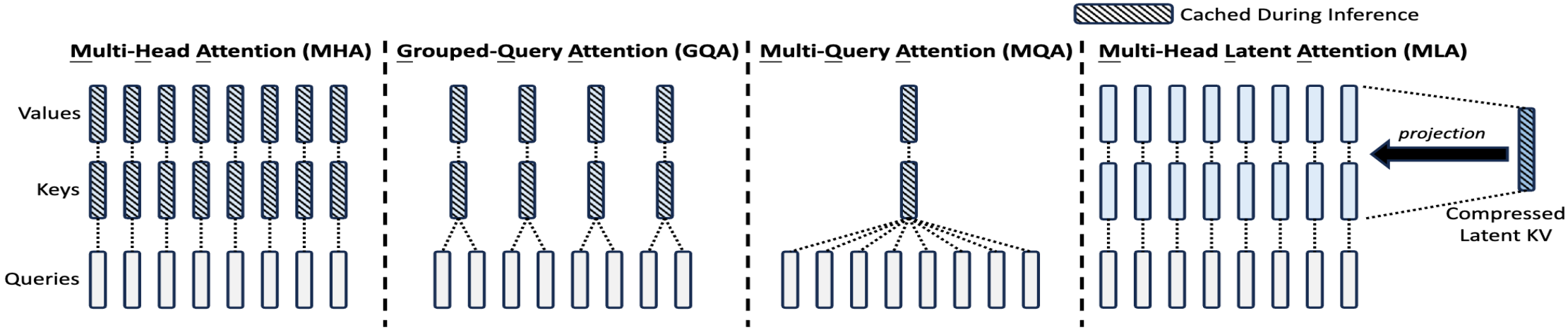
- 注意力存在**访存瓶颈**（KV缓存）
- GQA/MQA减少N倍缓存，但**性能损失**严重
- DeepSeek提出**MLA**

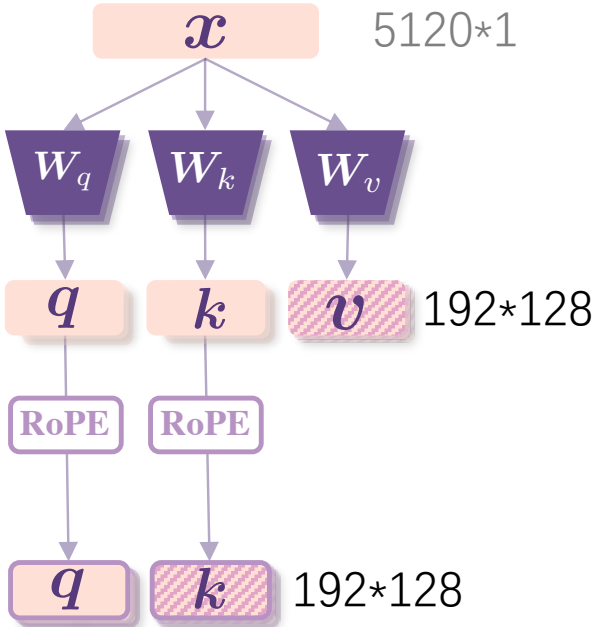
MHA vs. MLA

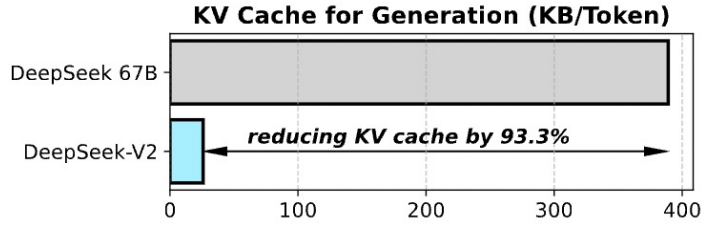


- 注意力存在**访存瓶颈**（KV缓存）
- GQA/MQA减少N倍缓存，但**性能损失**严重
- DeepSeek提出**MLA**

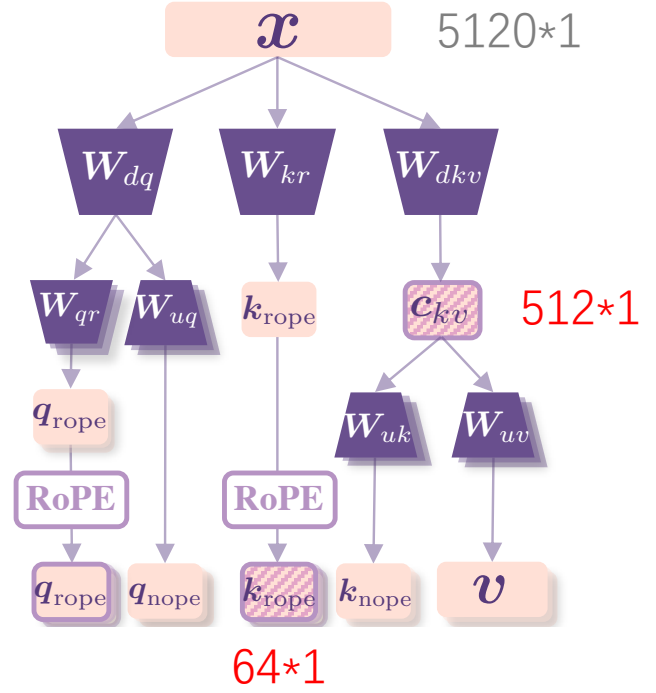
MHA vs. MLA







- 192*128*2 = 49152
- 64+512 = 576 (1.2%)



MHA vs. MLA

Benchmark (Metric)	# Shots	Dense 7B w/ MQA	Dense 7B w/ GQA (8 Groups)	Dense 7B w/ MHA
# Params	-	7.1B	6.9B	6.9B
BBH (EM)	3-shot	33.2	35.6	**37.0**
MMLU (Acc.)	5-shot	37.9	41.2	**45.2**
C-Eval (Acc.)	5-shot	30.0	37.7	**42.9**
CMMLU (Acc.)	5-shot	34.6	38.4	**43.5**
		-8.2	-3.9	

- GQA/MQA即使参数量对齐，**性能损失**仍然严重

MHA vs. MLA

Benchmark (Metric)	# Shots	Small MoE w/ MHA	Small MoE w/ MLA	Large MoE w/ MHA	Large MoE w/ MLA
# Activated Params	-	2.5B	2.4B	25.0B	21.5B
# Total Params	-	15.8B	15.7B	250.8B	247.4B
KV Cache per Token (# Element)	-	110.6K	15.6K	860.2K	34.6K
BBH (EM)	3-shot	37.9	**39.0**	46.6	**50.7**
MMLU (Acc.)	5-shot	48.7	**50.0**	57.5	**59.0**
C-Eval (Acc.)	5-shot	**51.6**	50.9	57.9	**59.2**
CMMLU (Acc.)	5-shot	52.3	**53.4**	60.7	**62.5**
			+0.7		+2.2

- MLA减少90%以上缓存，**性能持平或更优**

MHA vs. MLA

$$
\begin{aligned}
\boldsymbol{q}_i^{(h)} &= \boldsymbol{x}_i \boldsymbol{W}_q^{(h)} \\
\boldsymbol{k}_i^{(h)} &= \boldsymbol{x}_i \boldsymbol{W}_k^{(h)} \\
\boldsymbol{v}_i^{(h)} &= \boldsymbol{x}_i \boldsymbol{W}_v^{(h)} \\
\boldsymbol{q}_{i,\text{rope}}^{(h)} &= \text{RoPE}(\boldsymbol{q}_i^{(h)}) \\
\boldsymbol{k}_{i,\text{rope}}^{(h)} &= \text{RoPE}(\boldsymbol{k}_i^{(h)})
\end{aligned}
$$

$$
\boldsymbol{o}_i^{(h)} = \text{Softmax}\left(\boldsymbol{q}_{i,\text{rope}}^{(h)} \boldsymbol{k}_{\leq i,\text{rope}}^{(h)\top}\right) \boldsymbol{v}_{\leq i}^{(h)},
$$

$$
\text{MHA}(\boldsymbol{x}_i) = \left[\boldsymbol{o}_i^{(1)}, \ldots, \boldsymbol{o}_i^{(n_h)}\right] \boldsymbol{W}_o,
$$

$$
\begin{aligned}
\boldsymbol{q}_{i,\text{rope}}^{(h)}, \boldsymbol{k}_{i,\text{rope}} &= \text{RoPE}\left(\boldsymbol{x}_i \boldsymbol{W}_{dq} \boldsymbol{W}_{qr}^{(h)}, \boldsymbol{x}_i \boldsymbol{W}_{kr}\right), \\
\boldsymbol{q}_{i,\text{nope}}^{(h)} &= \boldsymbol{x}_i \boldsymbol{W}_{dq} \boldsymbol{W}_{qc}^{(h)}, \\
\boldsymbol{c}_{i,kv} &= \boldsymbol{x}_i \boldsymbol{W}_{dkv}, \\
\boldsymbol{k}_{i,\text{nope}}^{(h)}, \boldsymbol{v}_i^{(h)} &= \boldsymbol{c}_{i,kv} \boldsymbol{W}_{uk}^{(h)}, \boldsymbol{c}_{i,kv} \boldsymbol{W}_{uv}^{(h)},
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{o}_i^{(h)} = \text{Softmax}\Big(&\boldsymbol{q}_{i,\text{rope}}^{(h)} \boldsymbol{k}_{\leq i,\text{rope}}^{(h)\top} + \boldsymbol{q}_{i,\text{nope}} \boldsymbol{k}_{\leq i,\text{nope}}^{(h)\top}\Big) \\
&\cdot \boldsymbol{v}_{\leq i}^{(h)} \\
\text{MLA}(\boldsymbol{x}_i) = &\left[\boldsymbol{o}_i^{(1)}, \ldots, \boldsymbol{o}_i^{(n_h)}\right] \cdot \boldsymbol{W}_o. \qquad (3)
\end{aligned}
$$

MHA vs. MLA

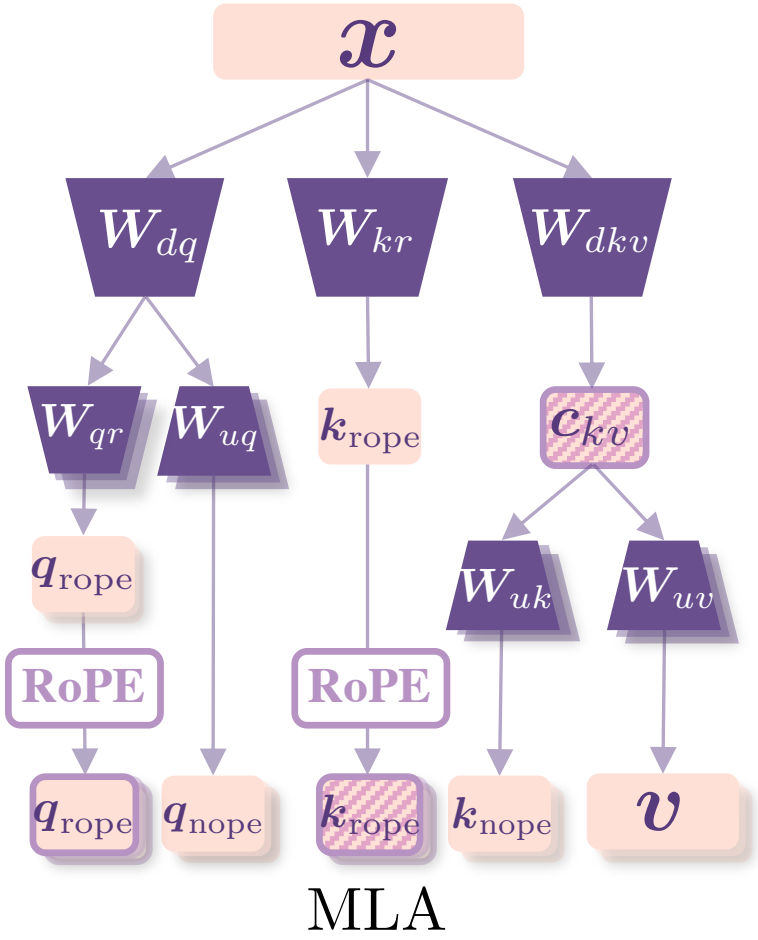
位置无关部分：矩阵吸收

$$\begin{aligned}\boldsymbol{q}_{i,\text{nope}}\boldsymbol{k}_{j,\text{nope}}^{\top} &= \left(\boldsymbol{x}_i\boldsymbol{W}_{dq}\boldsymbol{W}_{qc}\right)\left(\boldsymbol{c}_{j,kv}\boldsymbol{W}_{uk}\right)^{\top} \\ &= \boldsymbol{x}_i\left(\boldsymbol{W}_{dq}\boldsymbol{W}_{qc}\boldsymbol{W}_{uk}^{\top}\right)\boldsymbol{c}_{j,kv}^{\top}\end{aligned}$$

位置相关部分：正常计算

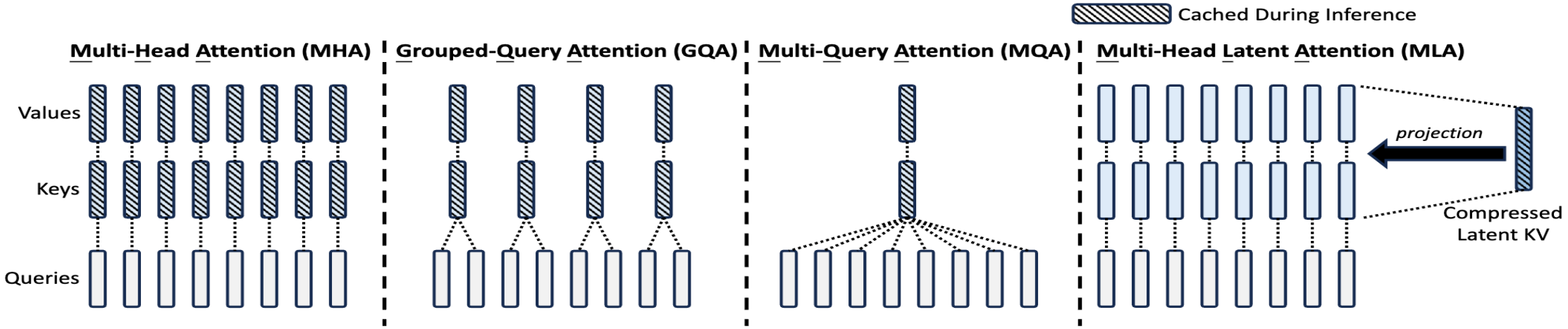
$$\begin{aligned}\boldsymbol{q}_{i,\text{rope}}\boldsymbol{k}_{j,\text{rope}}^{\top} &= \left(\boldsymbol{x}_i\boldsymbol{W}_{dq}\boldsymbol{W}_{qr}\boldsymbol{R}_i\right)\left(\boldsymbol{x}_j\boldsymbol{W}_{kr}\boldsymbol{R}_j\right)^{\top} \\ &= \boldsymbol{x}_i\left(\boldsymbol{W}_{dq}\boldsymbol{W}_{qc}\boldsymbol{R}_{j-i}\boldsymbol{W}_{kr}^{\top}\right)\boldsymbol{x}_j^{\top}\end{aligned}$$

无法吸收



FlashMLA

MHA vs. MLA



192

$$\boldsymbol{o}_i^{(h)} = \text{Softmax}\left(\boldsymbol{q}_{i,\text{rope}}^{(h)} \boldsymbol{k}_{\leq i,\text{rope}}^{(h)\top}\right) \boldsymbol{v}_{\leq i}^{(h)},$$

$$\text{MHA}(\boldsymbol{x}_i) = \left[\boldsymbol{o}_i^{(1)}, \ldots, \boldsymbol{o}_i^{(n_h)}\right] \boldsymbol{W}_o,$$

64 128

$$\boldsymbol{o}_i^{(h)} = \text{Softmax}\left(\boldsymbol{q}_{i,\text{rope}}^{(h)} \boldsymbol{k}_{\leq i,\text{rope}}^{(h)\top} + \boldsymbol{q}_{i,\text{nope}} \boldsymbol{k}_{\leq i,\text{nope}}^{(h)\top}\right) \cdot \boldsymbol{v}_{\leq i}^{(h)}$$

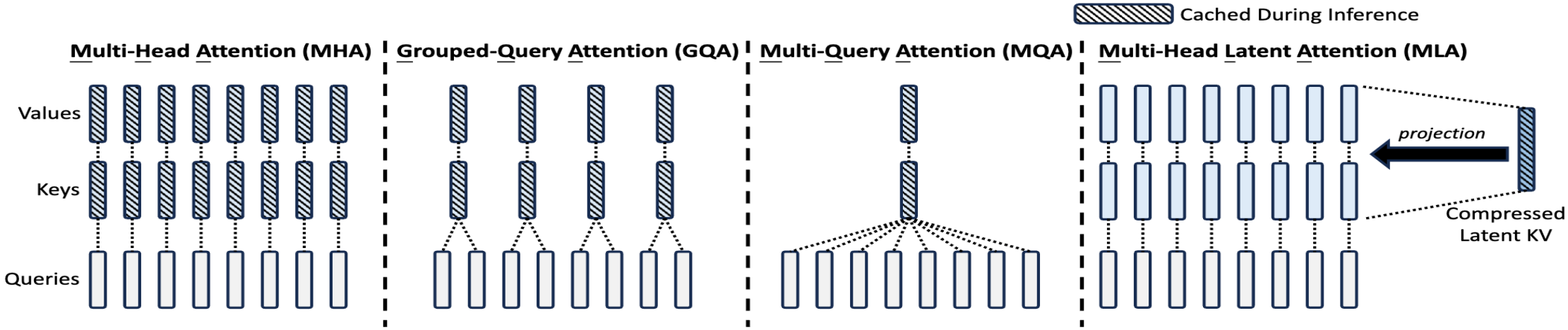
$$\text{MLA}(\boldsymbol{x}_i) = \left[\boldsymbol{o}_i^{(1)}, \ldots, \boldsymbol{o}_i^{(n_h)}\right] \cdot \boldsymbol{W}_o. \quad (3)$$

$$\boldsymbol{q}_{i,\text{nope}} \boldsymbol{k}_{j,\text{nope}}^{\top} = \left(\boldsymbol{x}_i \boldsymbol{W}_{dq} \boldsymbol{W}_{qc}\right) \left(\boldsymbol{c}_{j,kv} \boldsymbol{W}_{uk}\right)^{\top}$$

矩阵吸收

$$= \boldsymbol{x}_i \left(\boldsymbol{W}_{dq} \boldsymbol{W}_{qc} \boldsymbol{W}_{uk}^{\top}\right) \boldsymbol{c}_{j,kv}^{\top}$$

MHA vs. MLA



192

$$\boldsymbol{o}_i^{(h)} = \text{Softmax}\left(\boldsymbol{q}_{i,\text{rope}}^{(h)} \boldsymbol{k}_{\leq i,\text{rope}}^{(h)\top}\right) \boldsymbol{v}_{\leq i}^{(h)},$$

$$\text{MHA}(\boldsymbol{x}_i) = \left[\boldsymbol{o}_i^{(1)}, \ldots, \boldsymbol{o}_i^{(n_h)}\right] \boldsymbol{W}_o,$$

64 128

$$\boldsymbol{o}_i^{(h)} = \text{Softmax}\left(\boldsymbol{q}_{i,\text{rope}}^{(h)} \boldsymbol{k}_{\leq i,\text{rope}}^{(h)\top} + \boldsymbol{q}_{i,\text{nope}} \boldsymbol{k}_{\leq i,\text{nope}}^{(h)\top}\right) \cdot \boldsymbol{v}_{\leq i}^{(h)}$$

$$\text{MLA}(\boldsymbol{x}_i) = \left[\boldsymbol{o}_i^{(1)}, \ldots, \boldsymbol{o}_i^{(n_h)}\right] \cdot \boldsymbol{W}_o. \quad (3)$$

- Q[B, S, 128, 192] K[B, S, 128, 192]

- Qr[B, 1, 128, 64] Kr[B, S, 128, 64]
- Qc[B, 1, 128, 128] Kc[B, S, 128, 128]
- Qc[B, 1, 128, 512] Kc[B, S, 1, 512]

MHA2MLA

Poster

Q&A