CUDA实践下的 GPU并行计算原理与机制介绍

博士后： 纪泰

助教： 熊静飞、郭彬

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

自我介绍

https://taoji.eth.limo

- Efficient infrastructure via algorithm-hardware co-design
- Language-centric multimodal in-context learning

时间	单位	经历	导师
202309–202509	复旦大学	博士后	黄萱菁&邱锡鹏
201709–202306	华东师范大学 （硕博连读）	博士	吴苑斌&王晓玲
201309–201706	华东师范大学 （信息学保送）	学士	吴苑斌

熊静飞（研一）
复旦大学

郭彬（研一）
华东师范大学

Towards Economical Inference: Enabling DeepSeek's Multi-Head Latent Attention in Any Transformer-based LLMs

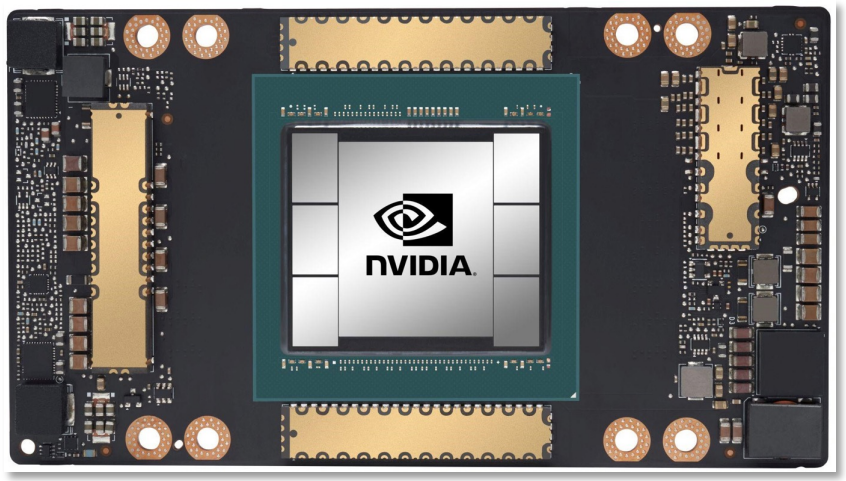
Tao Ji♠, Bin Guo♡, Yuanbin Wu♡,
Qipeng Guo◇, Lixing Shen♣, Zhan Chen♣, Xipeng Qiu♠, Qi Zhang♠, Tao Gui♠✉
♠Fudan University ♡East China Normal University ♣Hikvision Inc ◇Shanghai AI Lab
{taoji, tgui}@fudan.edu.cn {binguo@stu, ybwu@cs}.ecnu.edu.cn

MHA2MLA Public

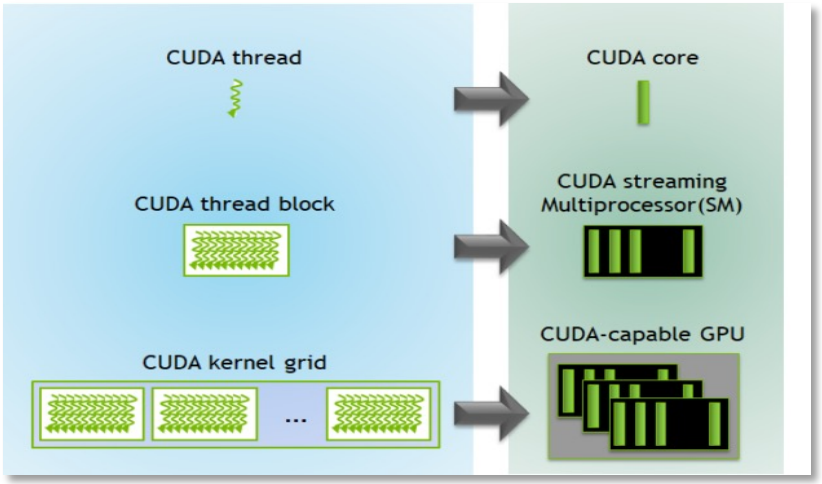
Towards Economical Inference: Enabling DeepSeek's Multi-Head Latent Attention in Any Transformer-based LLMs

Python ☆ 178 21

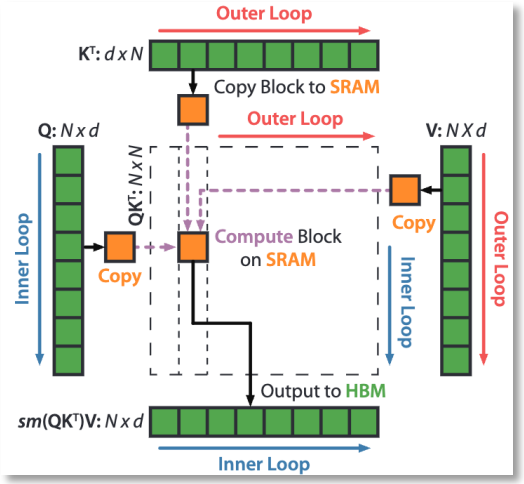
AI-Infra Seminar



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

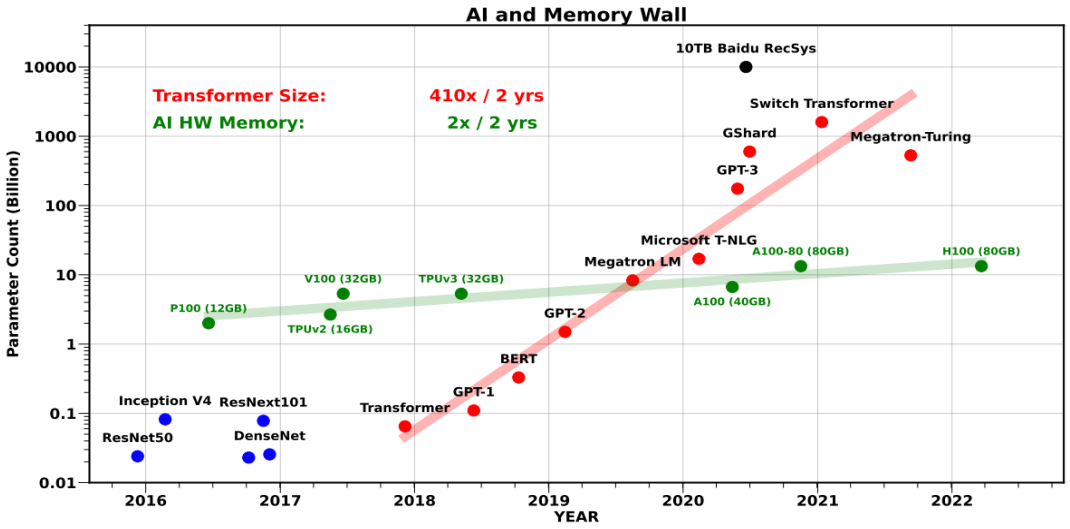
https://github.com/JT-Ushio/AI-Infra-Seminar

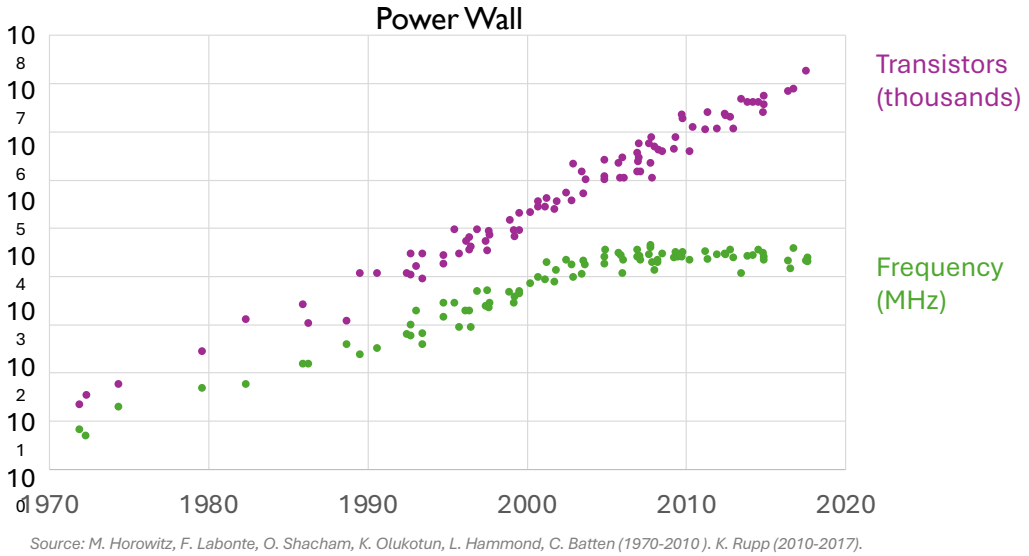
群聊：并行计算&CUDA讨论班

AI-Infrastructure

All *hardware* and *software* infrastructure used for model training and inference

- model scale ↑↑↑, GPU scale ↑
- Transformer hardware-friendly ⇨ computation-bottleneck

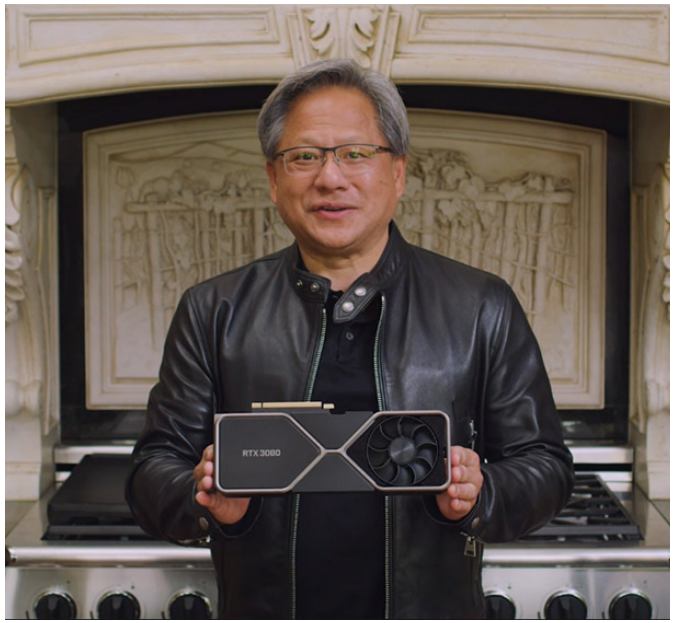




Source: M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten (1970-2010). K. Rupp (2010-2017).

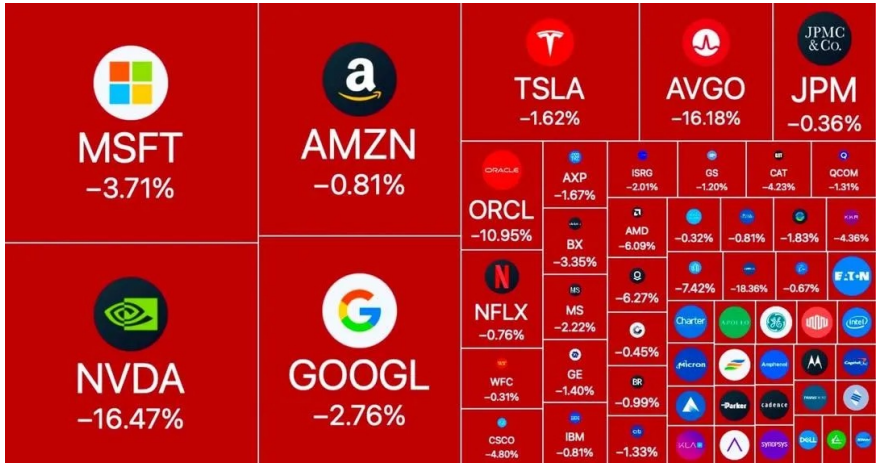
AI-Infrastructure

More GPU

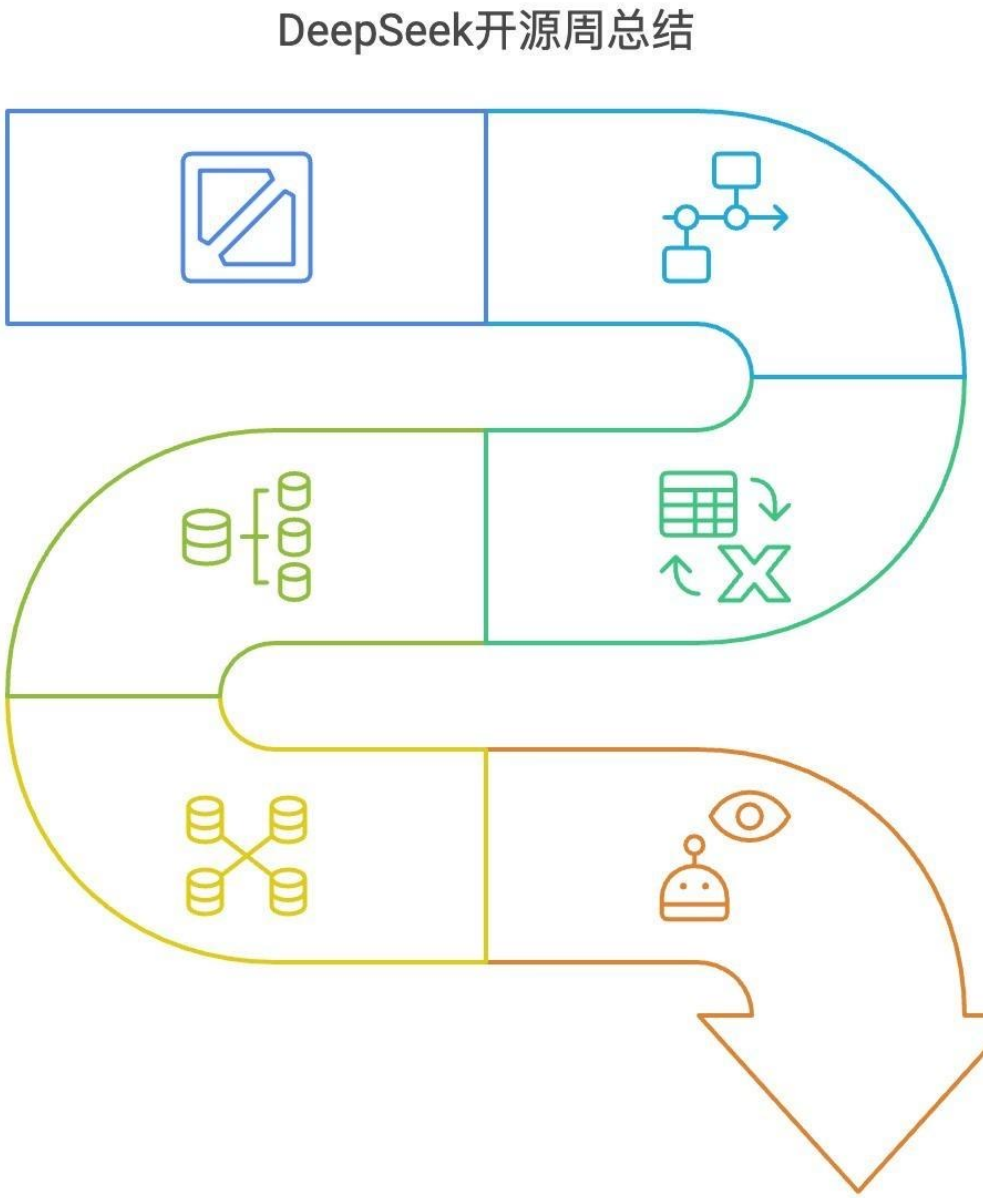


More efficient infrastructure

deepseek



AI-Infrastructure



AI-Infra Conferences

https://www.asplos-conference.org

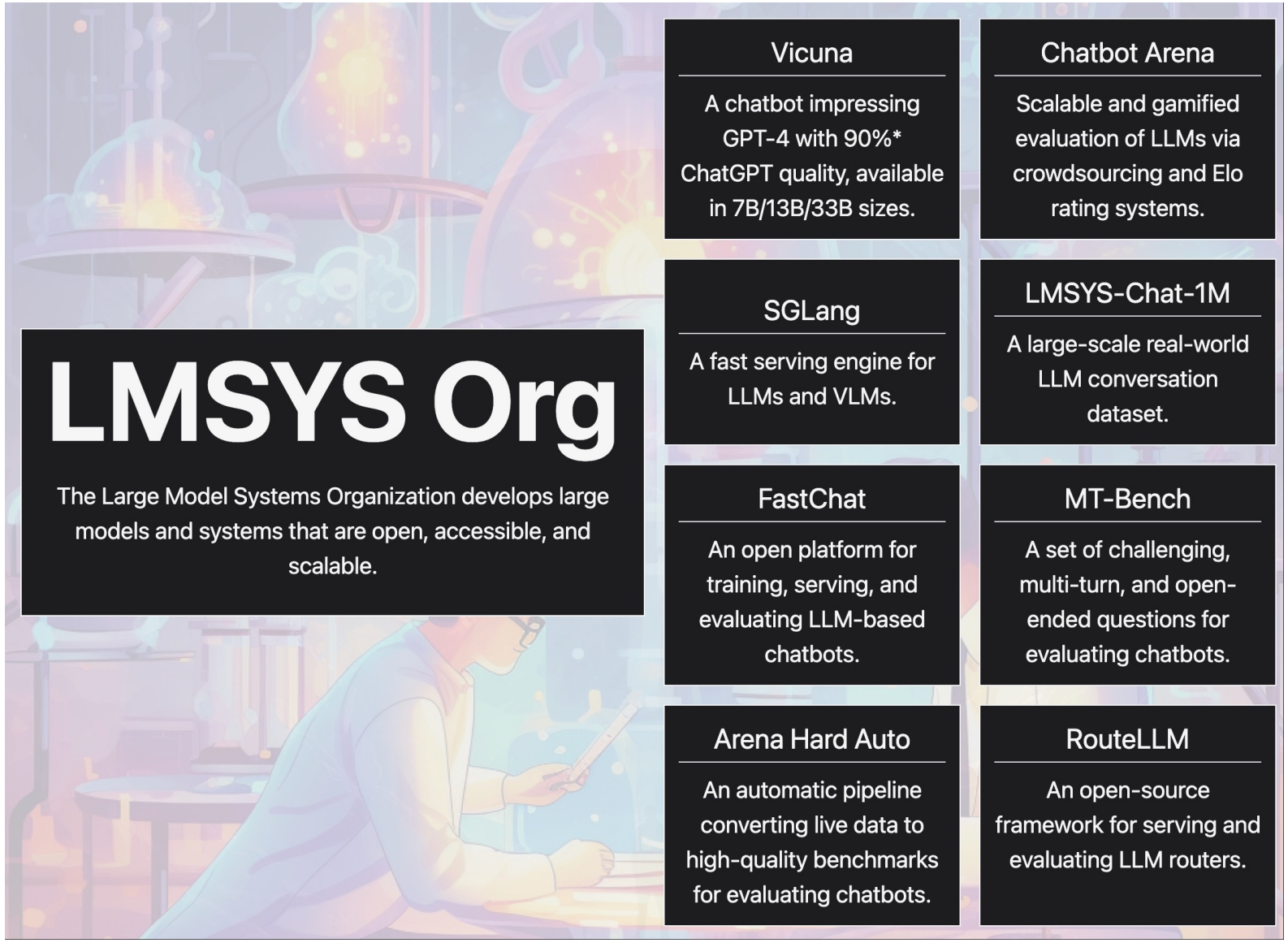
https://ppopp25.sigplan.org

https://mlsys.org

https://supercomputing.org

AI-Infra Research Team

Multi-university in 2023
UC Berkeley, Stanford, UCSD, CMU, MBZUAI
……



AI-Infra Research Team

catalyst

Home | People | Research | Publications

Catalyst is an interdisciplinary machine learning and systems research group exploring problems to automate learning systems. Our research spanning multiple layers of the machine learning and system stack. Our group is a collaboration between researchers from the Machine Learning Department, Computer Science Department and Electrical & Computer Engineering Department at the Carnegie Mellon University.

Research

XGrammar
Efficient, Flexible and Portable LLM Structured Generation

Read more »

FlexFlow Serve
Low-Latency, High-Performance LLM Serving

Read more »

Hyperband and ASHA
Principled early-stopping approaches for hyperparameter optimization

Read more »

TidalDecode
A sparse attention framework for large language model decoding

Read more »

Cortex
End-to-end compilation of ML applications with dynamic and irregular control flow and data structure accesses

Read more »

FlexFlow
Automatically Discovering Fast Parallelization Strategies for DNN Training

Read more »

Machine Learning Compilation for Large Language Models

Apache TVM Stack
TVM: An Automated End-to-End

TASO
The Tensor Algebra SuperOptimizer for Deep Learning

Faculty

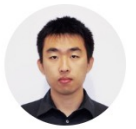

Beidi Chen
Assistant Professor


Tianqi Chen
Assistant Professor


Tim Dettmers
Assistant Professor


Greg Ganger
Professor


Phillip B. Gibbons
Professor


Zhihao Jia
Assistant Professor


Todd C. Mowry
Professor


Ameet Talwalkar
Assistant Professor


Eric Xing
Professor

AI-Infra Research Team

AI-Infra Research Team

Tri Dao

Assistant Professor of Computer Science at Princeton University.
Chief Scientist at Together AI.

CV (updated 06/2025)

Previously: PhD, Department of Computer Science, Stanford University

tri [at] tridao (dot) me

Research Interests

Machine learning and systems, with a focus on efficient training and inference:

- Hardware-aware algorithms.
- Sequence models with long-range memory.

Current PhD Students

- Ted Zadouri
- Berlin Chen
- Wentao Guo

AI-Infra Research Team

IPADS | 并行与分布式系统研究所

教师们


陈海波
教授


臧斌宇
教授


陈榕
教授


夏虞斌
教授


王肇国
长聘副教授

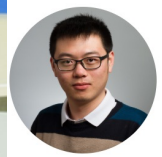

糜泽羽
副教授


吴明瑜
副教授


华志超
副教授


古金宇
副教授

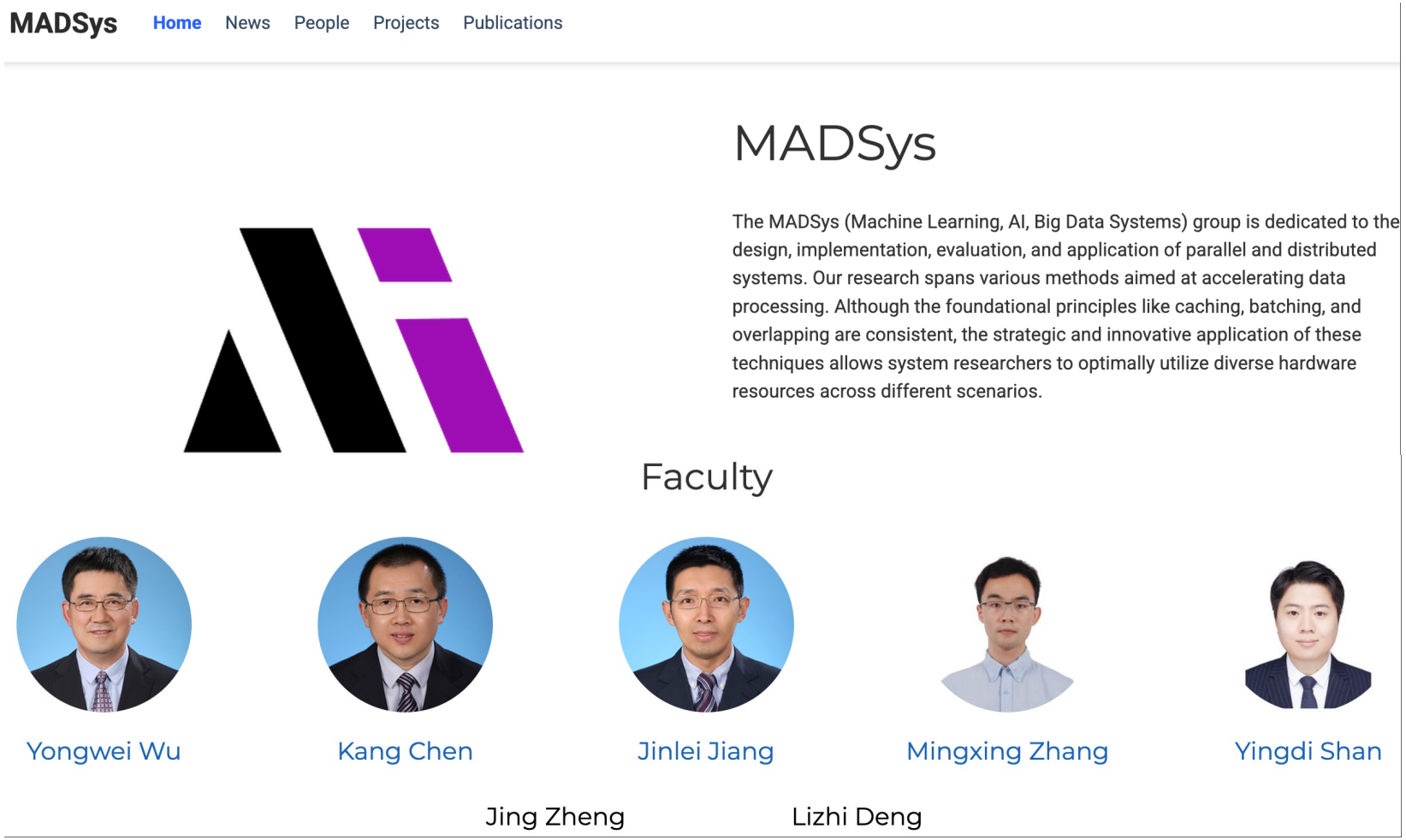

魏星达
助理教授


董明凯
副教授

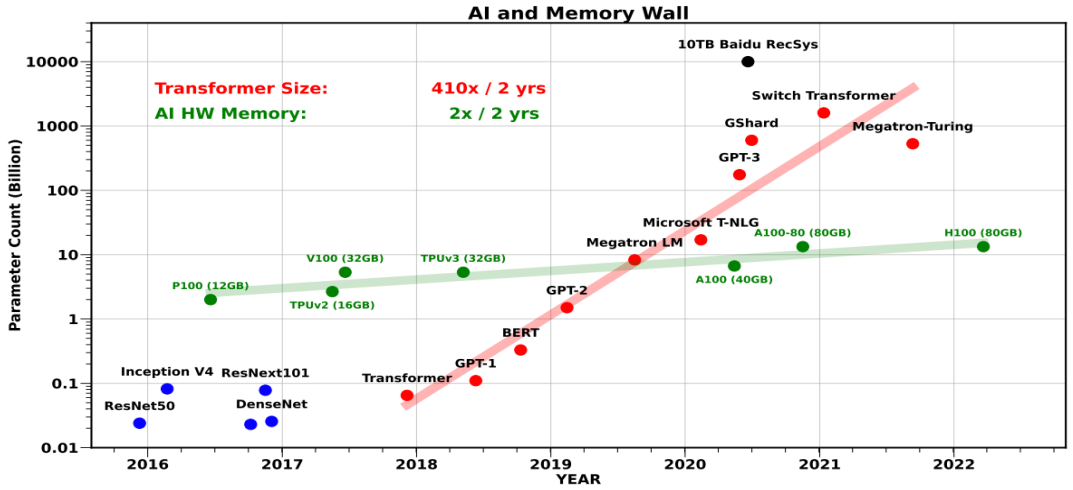

杜东
助理研究员


余炀
助理研究员

AI-Infra Research Team



AI-Infra Research Team



CUDA

⇩

Cutlass

⇩

Cutlass Python DSL

⇩

Triton & Cutile

► more important

► easier

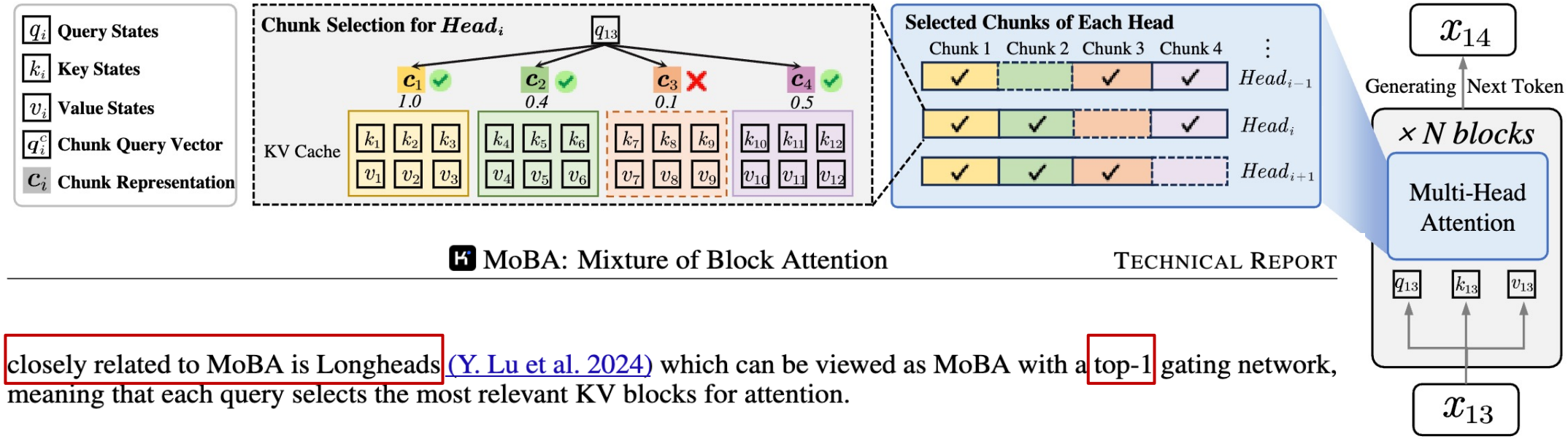
System → AI/LLM🤝Sys ← AI/LLM

Why Am I Interested?

LONGHEADS: Multi-Head Attention is Secretly a Long Context Processor

Yi Lu[1*], Xin Zhou[1*], Wei He[1], Jun Zhao[1],
Tao Ji[1†], Tao Gui[2†], Qi Zhang[1†], Xuanjing Huang[1,3]

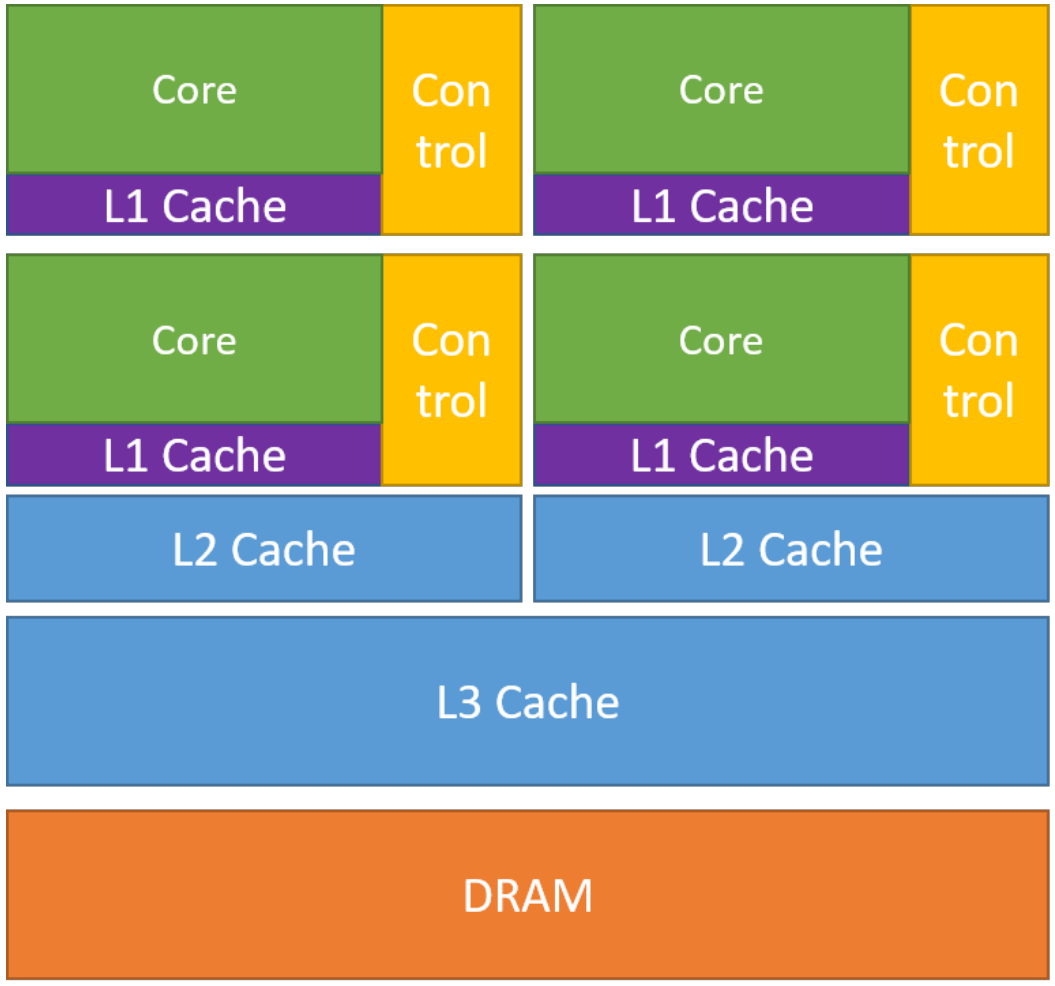
上下文分块+稀疏选择



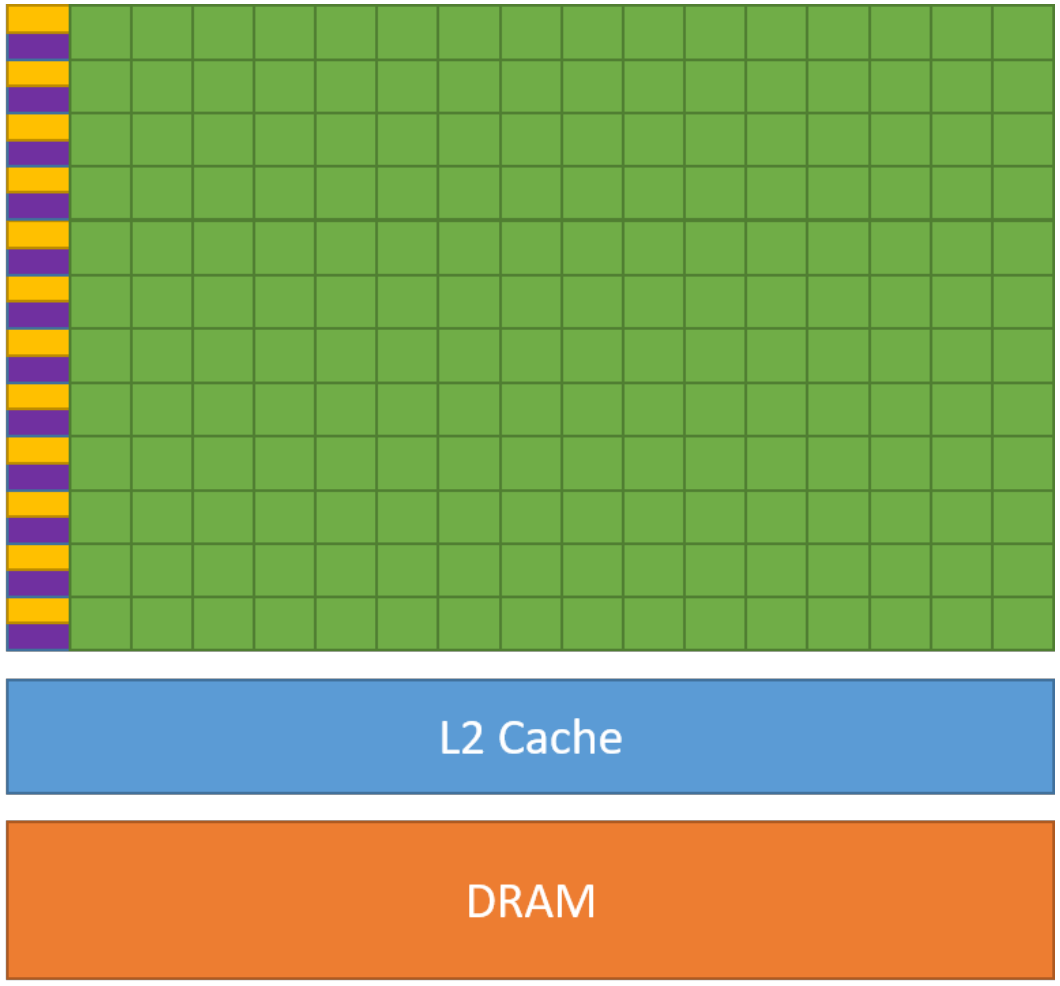
closely related to MoBA is Longheads (Y. Lu et al. 2024) which can be viewed as MoBA with a top-1 gating network, meaning that each query selects the most relevant KV blocks for attention.

4/3/2.5 4(meta) but findings 🤦

GPU vs. CPU

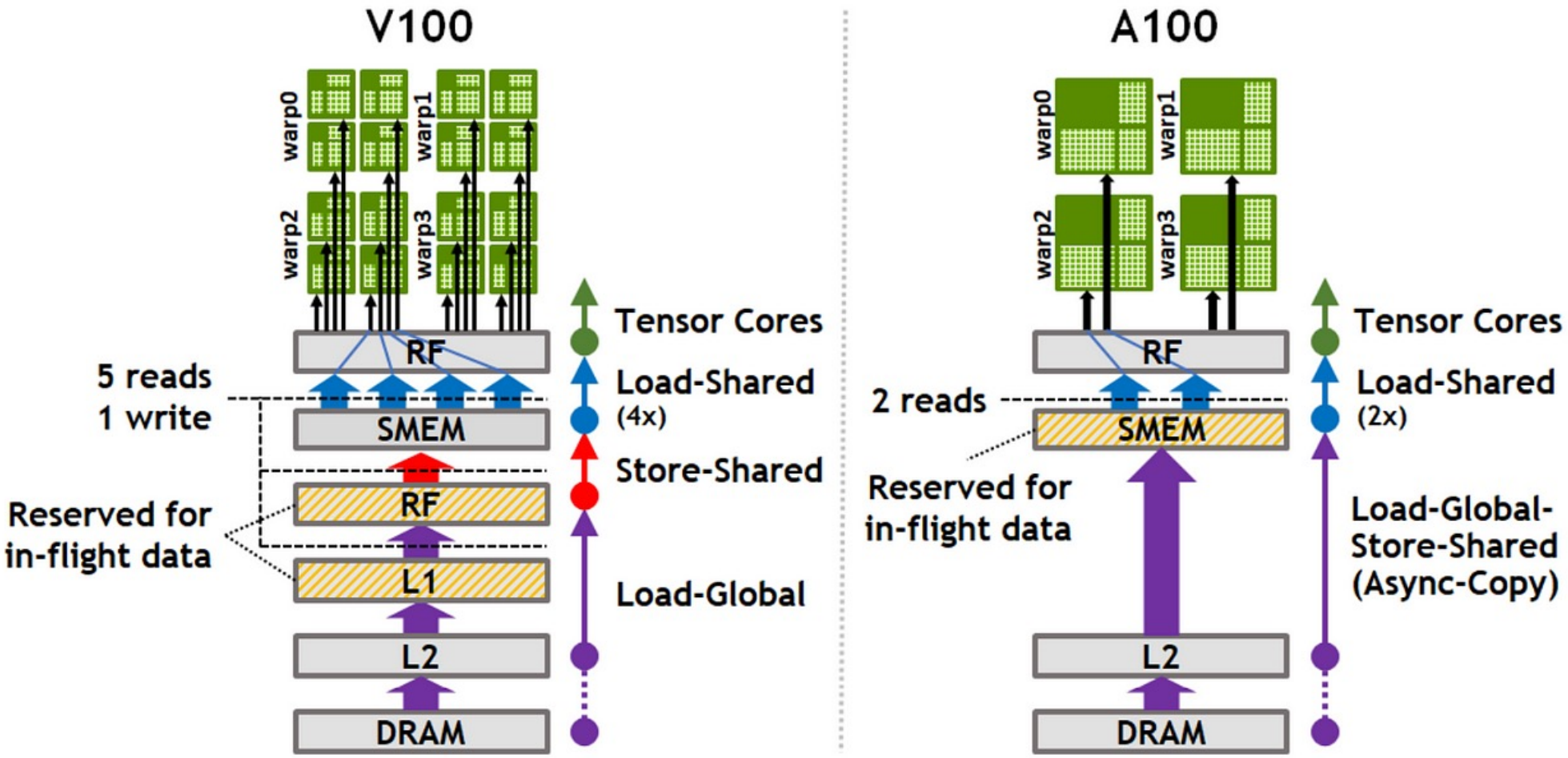


CPU

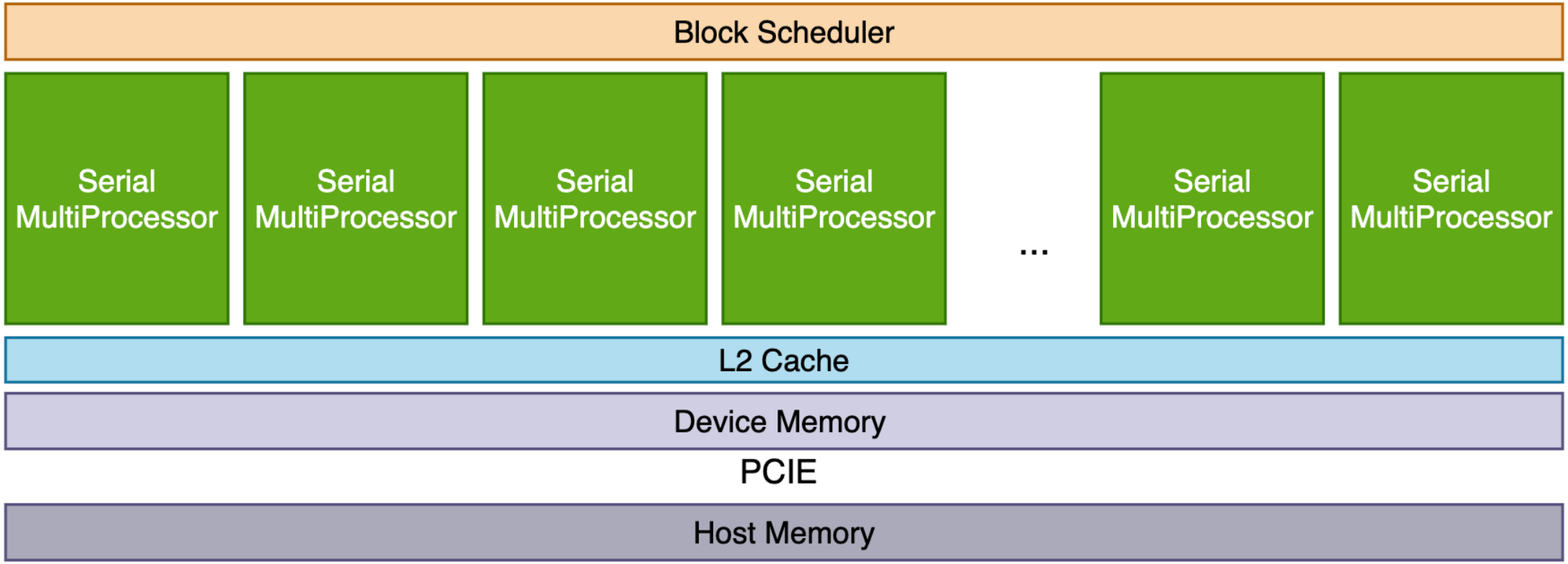


GPU

GPU


V100
warp0
warp1
warp2
warp3
RF
SMEM
RF
L1
L2
DRAM
5 reads
1 write
Reserved for
in-flight data
Tensor Cores
Load-Shared
(4x)
Store-Shared
Load-Global
A100
warp0
warp1
warp2
warp3
RF
SMEM
L2
DRAM
2 reads
Reserved for
in-flight data
Tensor Cores
Load-Shared
(2x)
Load-Global-
Store-Shared
(Async-Copy)

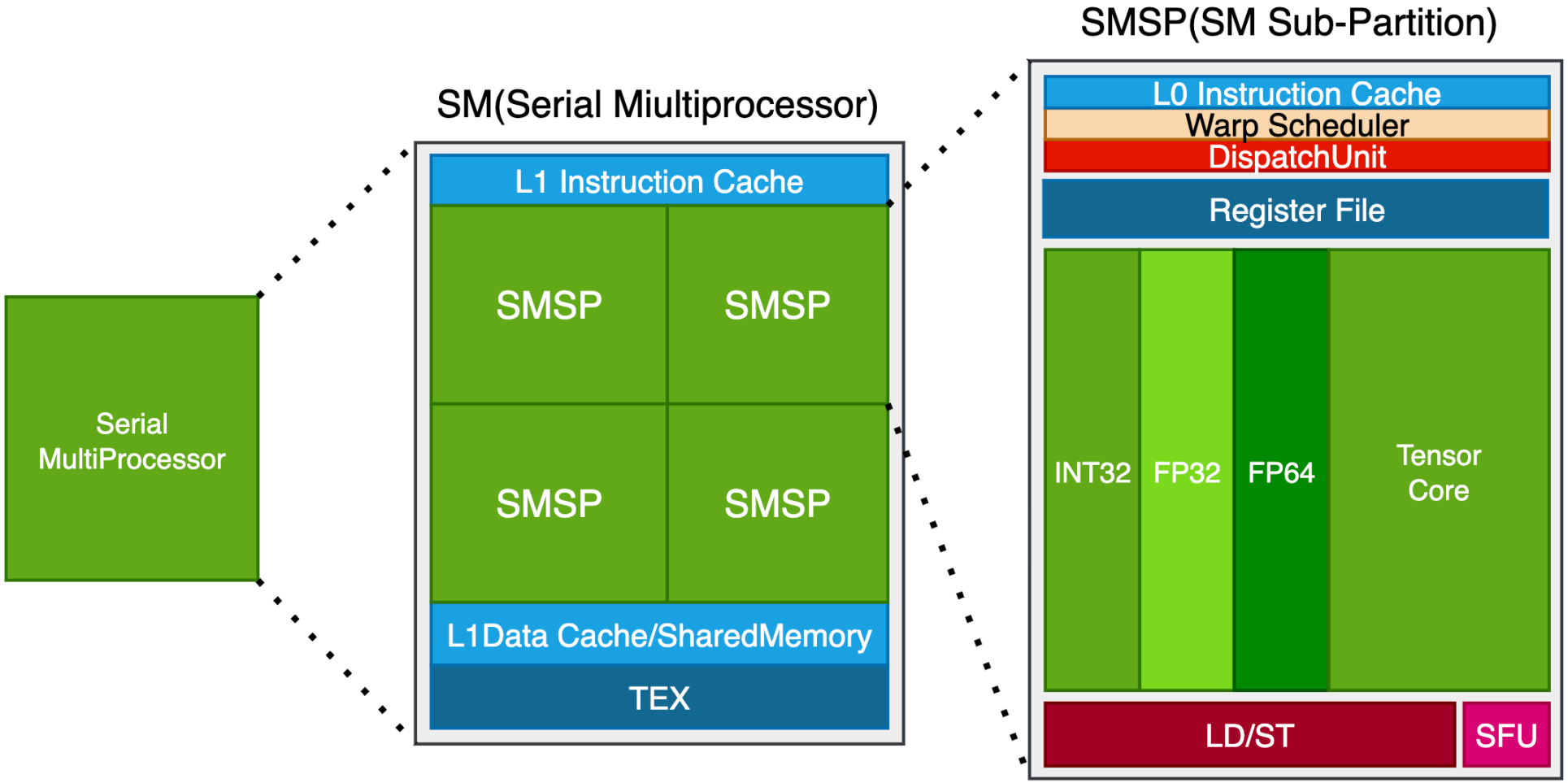
GPU


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

GPU


PCI Express 4.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
L2 Cache
L2 Cache
Memory Controller
HBM2
High-Speed Hub
NVLink

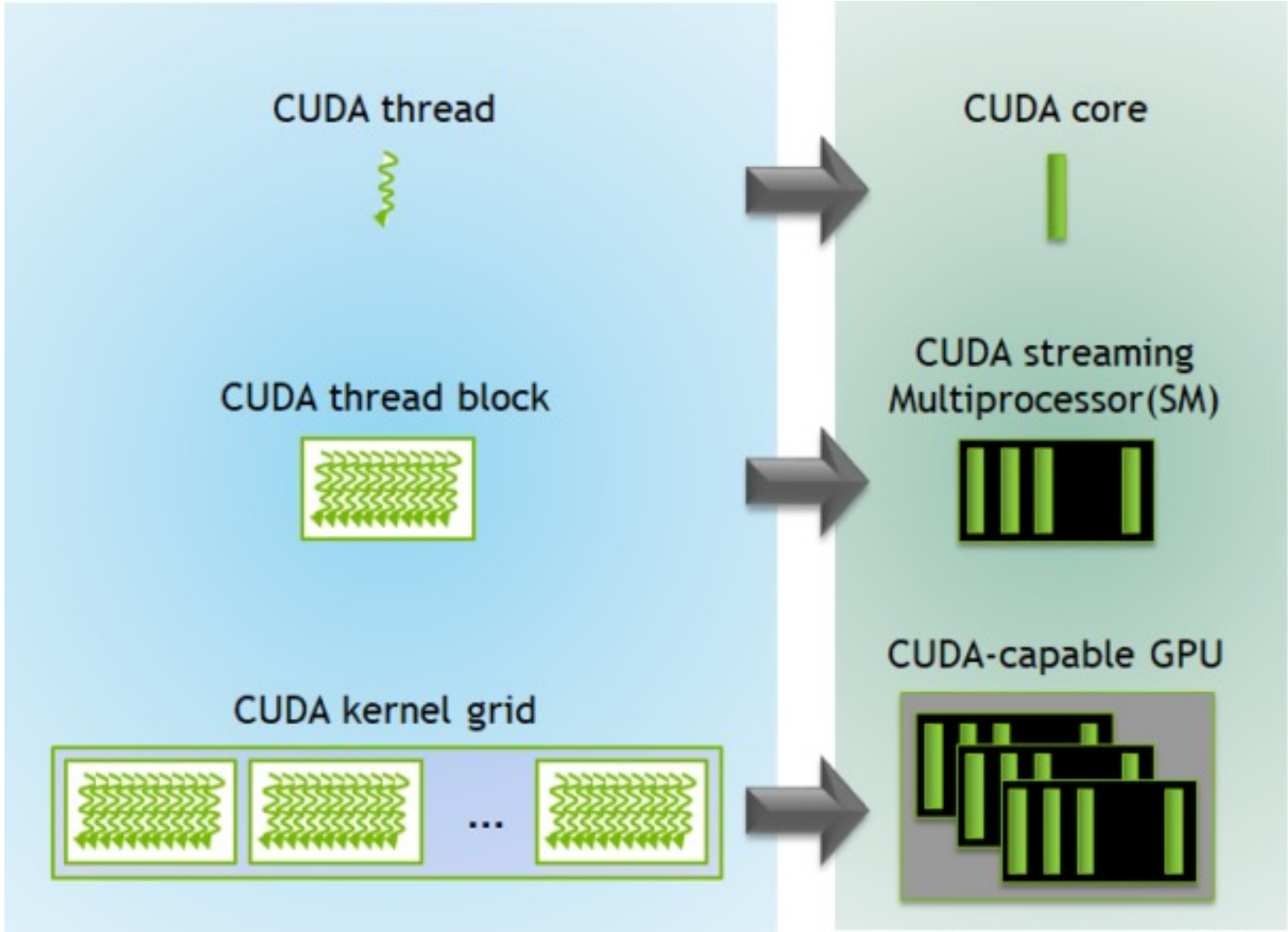
GPU


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

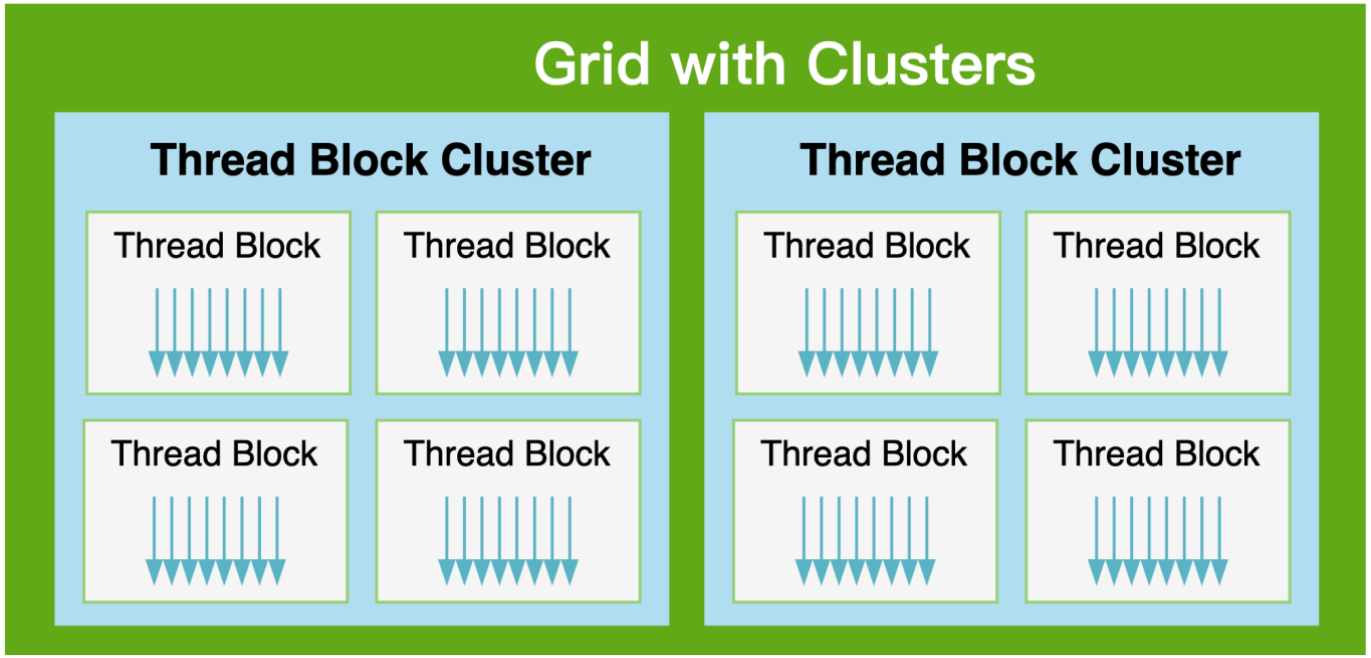
GPU


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

CUDA Programming



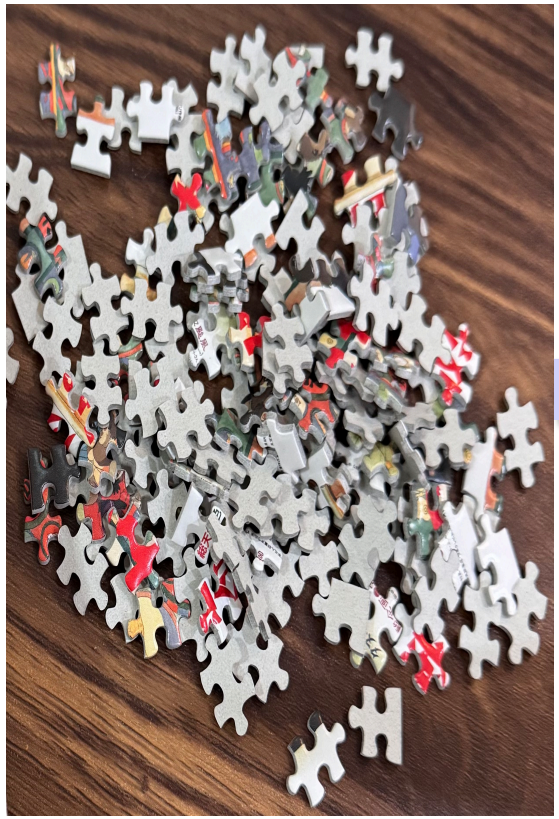
CUDA Programming



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

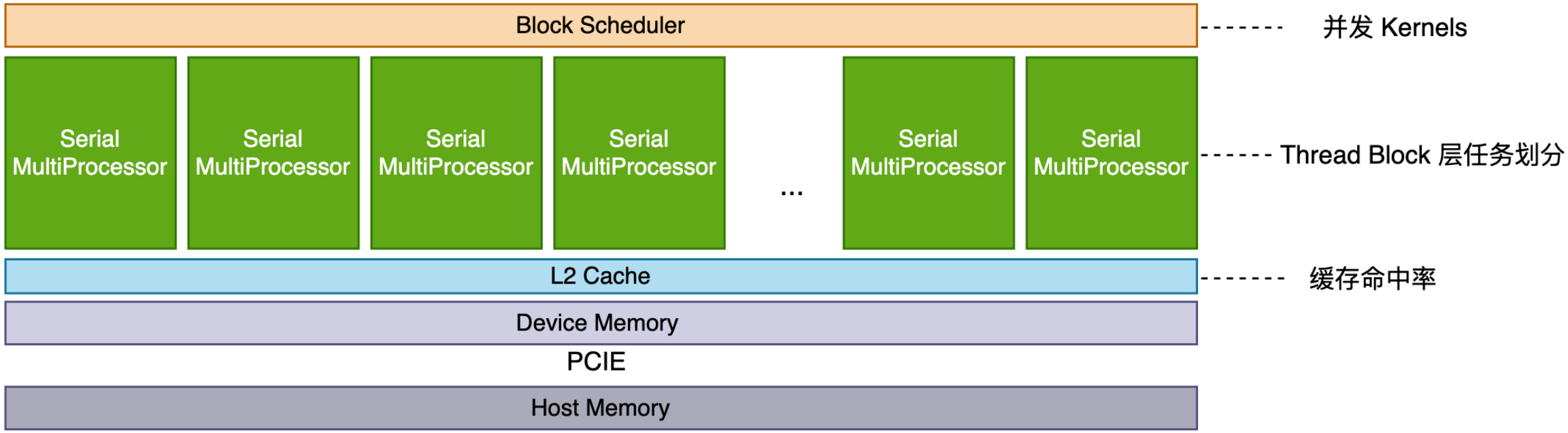
- **Thread-level Programming**
- **Single instruction, multiple threads (SIMT)**

CUDA Programming: SIMT



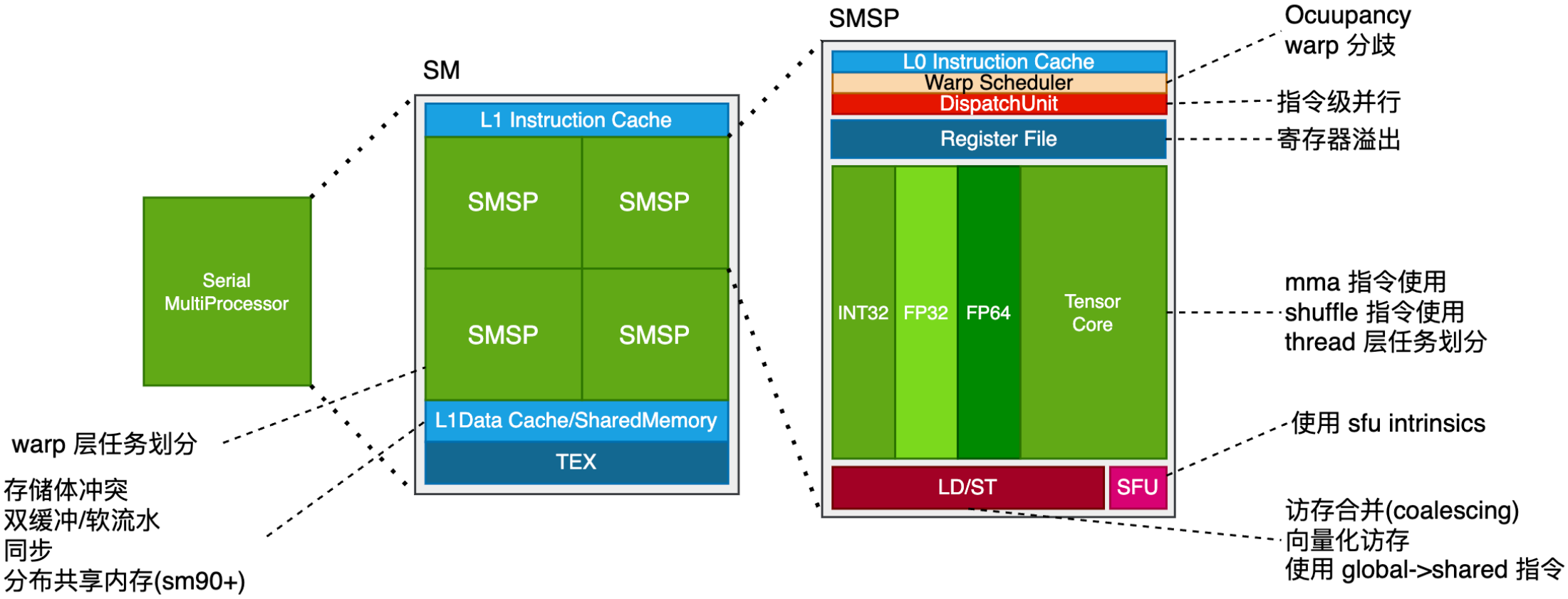
- ID
- R, C=ID // Col, ID % Col
- 🧩

CUDA Programming



outside an SM

CUDA Programming



within an SM

C++: Hello World!

hello_world.py

```python
import torch
from torch.utils.cpp_extension import load_inline

cpp_source = """
std::string hello_world() {
    return "Hello World!";
}
"""

module = load_inline(
    name='moss_op',
    cpp_sources=[cpp_source],
    functions=['hello_world'],
    verbose=True,
)

print(module.hello_world())
```

`load_inline`

CUDA🤝Pytorch

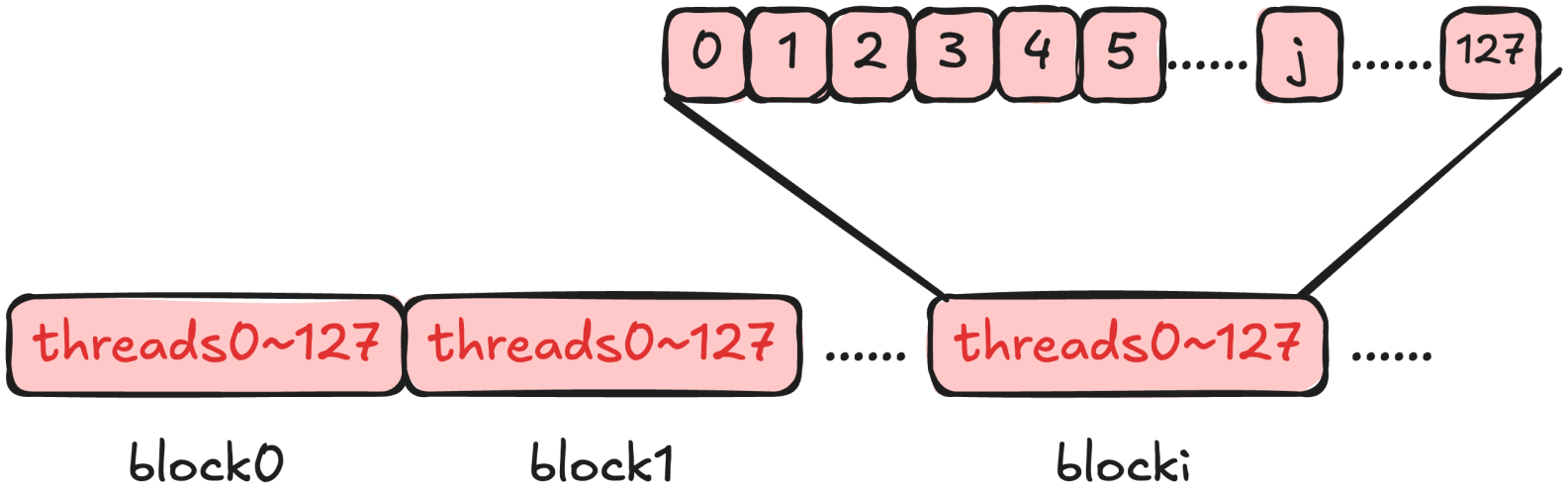
load_inline_cuda.py

```python
import torch
from torch.utils.cpp_extension import load_inline

cuda_source = '''
__global__ void square_vector_kernel(const float* vector, float* result, int col) {
    int c = blockIdx.x * blockDim.x + threadIdx.x;
```

修饰符	调用者	执行位置	典型用途
`__global__`	host	device	GPU kernel，从 CPU 发起
`__device__`	device	device	GPU 上运行的辅助函数
`__host__`	host	host	CPU 上的普通函数（默认）

```python
    return result;
}
'''

cpp_source = "torch::Tensor square_vector(torch::Tensor vector);"

module = load_inline(
    name='moss_op',
    cpp_sources=cpp_source,
    cuda_sources=cuda_source,
    functions=['square_vector'],
)

a = torch.tensor([1., 2., 3.], device='cuda')
print(module.square_vector(a))
```

CUDA🤝Pytorch



i*128+j

Profiler

cuda_profile.py

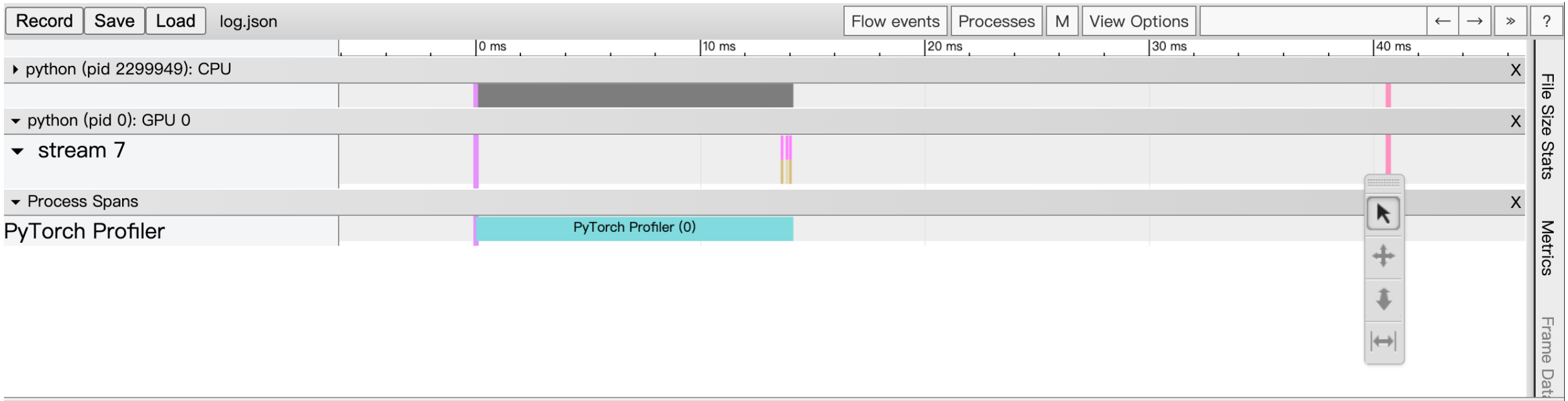
```python
with profile(
    activities=[ProfilerActivity.CUDA],
    schedule=torch.profiler.schedule(wait=2, warmup=2, active=3, repeat=1),
    on_trace_ready=torch.profiler.tensorboard_trace_handler('./log/moss_op'),
    record_shapes=True,
    with_stack=True,
    with_flops=True
) as prof:
    for i in range(7):
        with record_function("square_vector_custom"):
            module.square_vector(a)
        prof.step()

print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

```
tensor([9.7066e-01, 6.9254e-02, 5.9329e-03,  ..., 1.6538e-03, 3.4866e-05,
        2.4144e-01], device='cuda:0')
```

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg	# of Calls
square_vector_kernel(float const*, float*, int)	0.00%	0.000us	0.00%	0.000us	0.000us	364.991us	100.00%	364.991us	121.664us	3
cudaLaunchKernel	24.99%	35.837us	24.99%	35.837us	11.946us	0.000us	0.00%	0.000us	0.000us	3
cudaDeviceSynchronize	75.01%	107.547us	75.01%	107.547us	107.547us	0.000us	0.00%	0.000us	0.000us	1

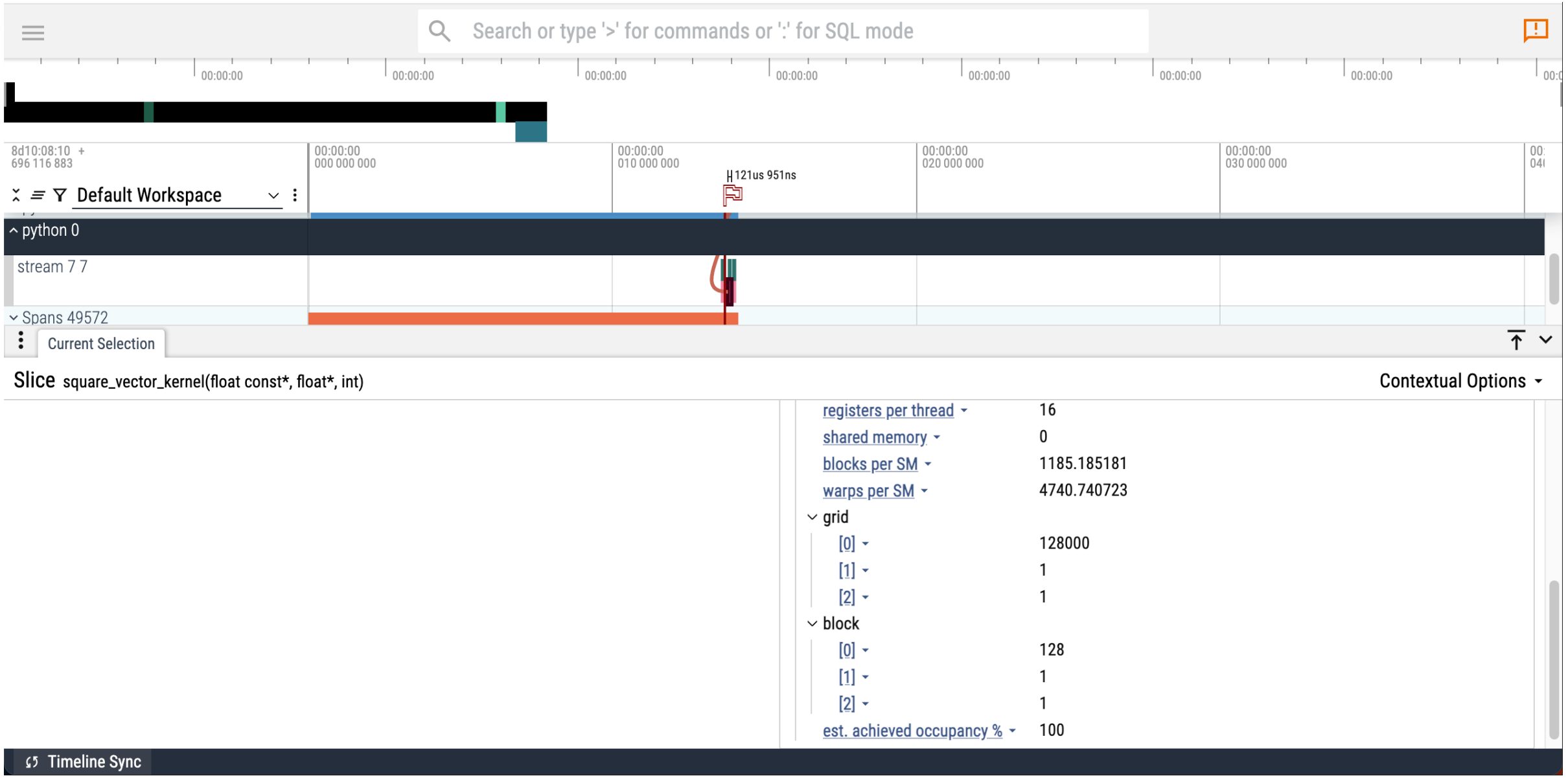
```
Self CPU time total: 143.384us
Self CUDA time total: 364.991us
```

Profiler



context	1
stream	7
correlation	32
registers per thread	16
shared memory	0
blocks per SM	1185.185181
warps per SM	4740.740723
grid	[128000, 1, 1]
block	[128, 1, 1]
est. achieved occupancy %	100

Profiler


Search or type '>' for commands or ':' for SQL mode
00:00:00
8d10:08:10 +
696 116 883
000 000 000
010 000 000
121us 951ns
020 000 000
030 000 000
Default Workspace
python 0
stream 7 7
Spans 49572
Current Selection
Slice square_vector_kernel(float const*, float*, int)
Contextual Options
registers per thread 16
shared memory 0
blocks per SM 1185.185181
warps per SM 4740.740723
grid
[0] 128000
[1] 1
[2] 1
block
[0] 128
[1] 1
[2] 1
est. achieved occupancy % 100
Timeline Sync

Profiler

作业一: Profiler LLMs Module

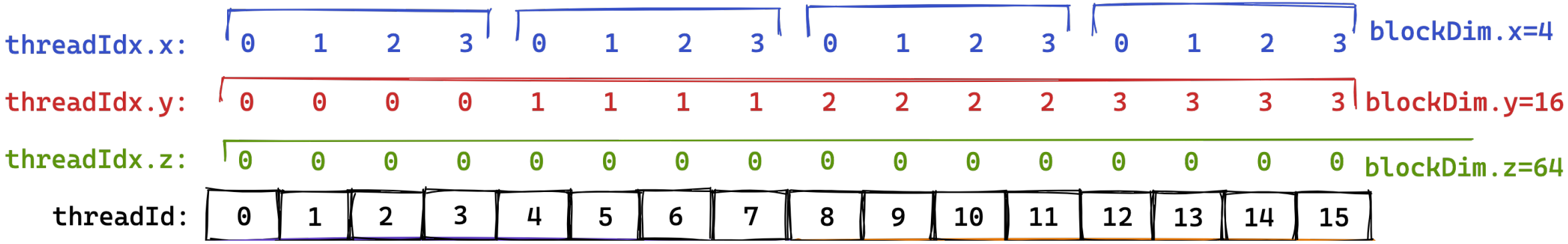
- Attention
- LayerNorm
- Feedforward
- LMhead

……

2D/3D Block

blocks.cu

```
dim3 threads_per_block(4, 16, 64);
dim3 blocks_per_grid(cdiv(N, threads_per_block.x), cdiv(M, threads_per_block.y), cdiv(K, threads_per_block.z));
```



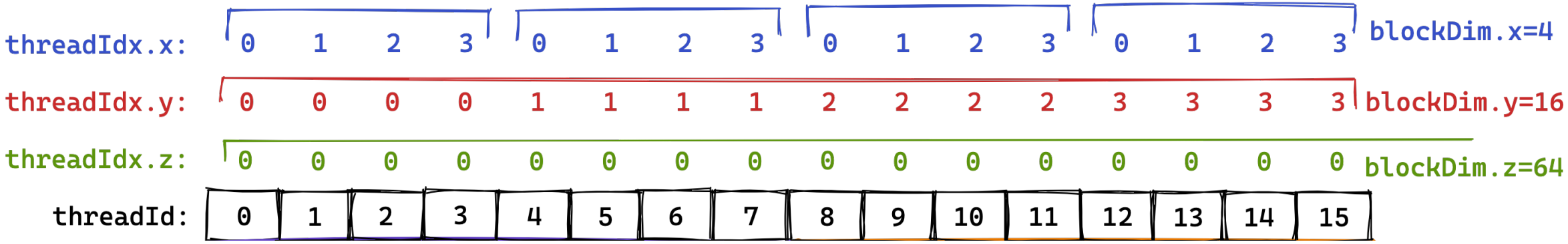
$$threadId = threadIdx.x + blockDim.x * threadIdx.y + blockDim.x * blockDim.y * threadIdx.z$$

4*16*64=4096

2D/3D Block

blocks.cu

```
dim3 threads_per_block(4, 16, 64);
dim3 blocks_per_grid(cdiv(N, threads_per_block.x), cdiv(M, threads_per_block.y), cdiv(K, threads_per_block.z));
```



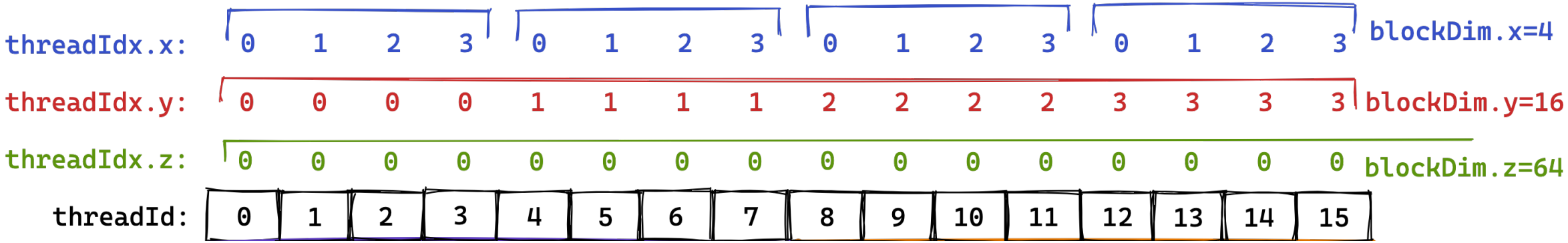
限制类型	数值	说明
每个 block 最多线程数	`1024`	即：`blockDim.x * blockDim.y * blockDim.z <= 1024`

每个维度的最大线程数	`blockDim.x <= 1024`
	`blockDim.y <= 1024`
	`blockDim.z <= 64`
		各维度上限依 GPU 架构而定（较常见值）

2D/3D Block

blocks.cu

```
dim3 threads_per_block(4, 16, 64);
dim3 blocks_per_grid(cdiv(N, threads_per_block.x), cdiv(M, threads_per_block.y), cdiv(K, threads_per_block.z));
```



check_device.py

```
import torch

prop = torch.cuda.get_device_properties(0)

print("Device name:", prop.name)
print("Max threads per block:", prop.max_threads_per_block)
print("Max threads per SM:", prop.max_threads_per_multi_processor)
print("Warp size:", prop.warp_size)
print("Number of SMs:", prop.multi_processor_count)
```

SiLU activation

warp_coalesced.py

```python
rows, cols = 8192, 8192
x = torch.randn((rows, cols), device="cuda")

with profile(
    activities=[ProfilerActivity.CUDA],
    schedule=torch.profiler.schedule(wait=1, warmup=1, active=2),
    on_trace_ready=torch.profiler.tensorboard_trace_handler("./log/warp_coalesced"),
    record_shapes=False,
    with_stack=False,
) as prof:
    for _ in range(4):
        with record_function("silu_torch"), torch.no_grad():
            out0 = torch.nn.functional.silu(x)
        with record_function("silu_coalesced"):
            out1 = module.silu_coalesced(x)
        with record_function("silu_uncoalesced"):
            out2 = module.silu_uncoalesced(x)
        prof.step()

assert torch.allclose(out1, out0) and torch.allclose(out2, out0)
print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

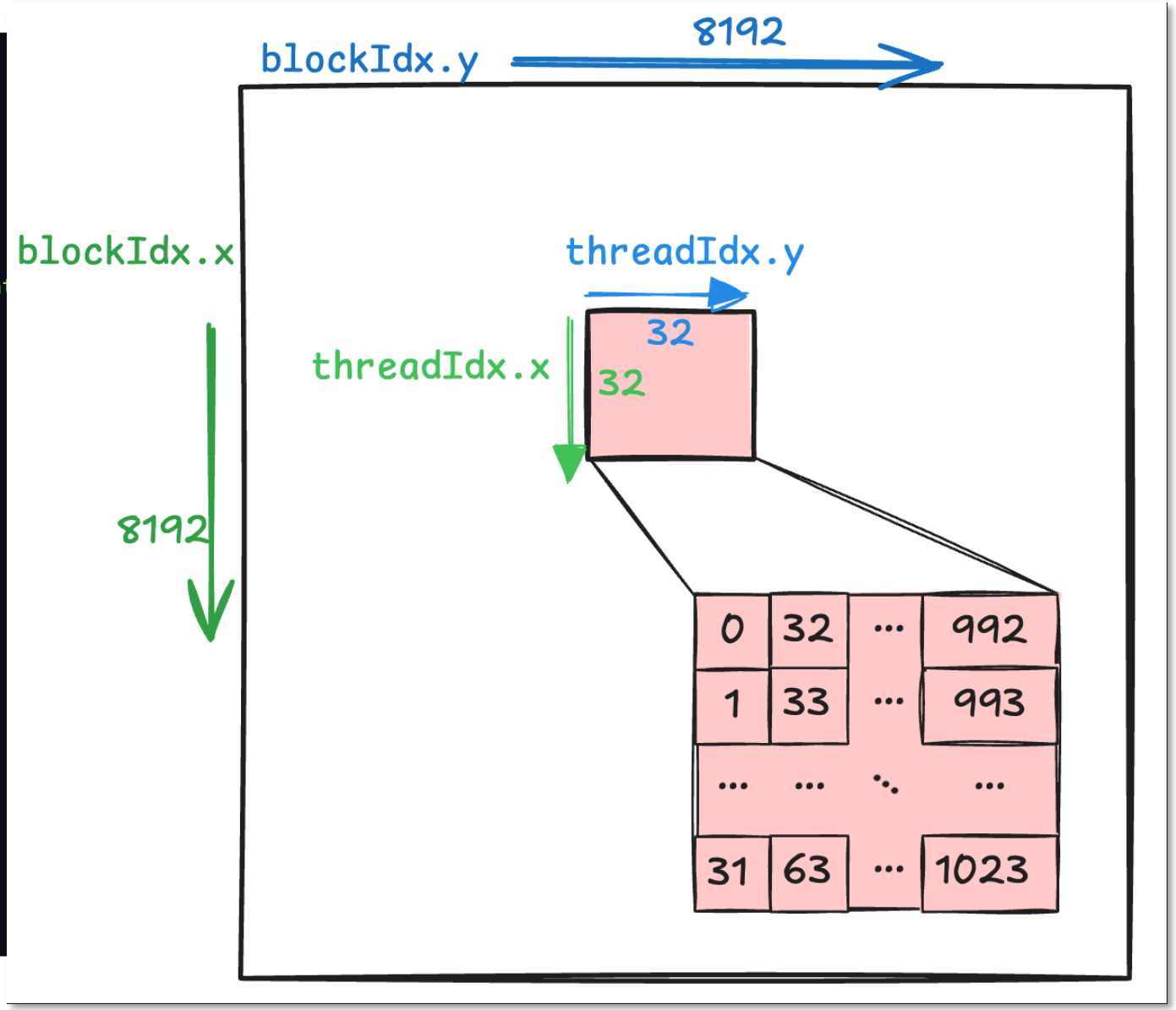
SiLU activation

```
Loading extension module moss_op...
-------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
                                                   Name    Self CPU %     Self CPU   CPU total %     CPU total  CPU time avg     Self CUDA  Self CUDA %    CUDA total  CUDA time avg
-------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
silu_uncoalesced_kernel(float const*, float*, int, i...       0.00%       0.000us       0.00%       0.000us       0.000us       7.175ms        81.10%       7.175ms       2.392ms
silu_coalesced_kernel(float const*, float*, int, int...       0.00%       0.000us       0.00%       0.000us       0.000us       1.067ms        12.06%       1.067ms     533.326us
void at::native::vectorized_elementwise_kernel<4, at...       0.00%       0.000us       0.00%       0.000us       0.000us     605.246us         6.84%     605.246us     302.623us
                                          cudaLaunchKernel       1.15%      69.085us       1.15%      69.085us      11.514us       0.000us         0.00%       0.000us       0.000us
                                     cudaDeviceSynchronize      98.85%       5.954ms      98.85%       5.954ms       5.954ms       0.000us         0.00%       0.000us       0.000us
-------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 6.024ms
Self CUDA time total: 8.847ms
```

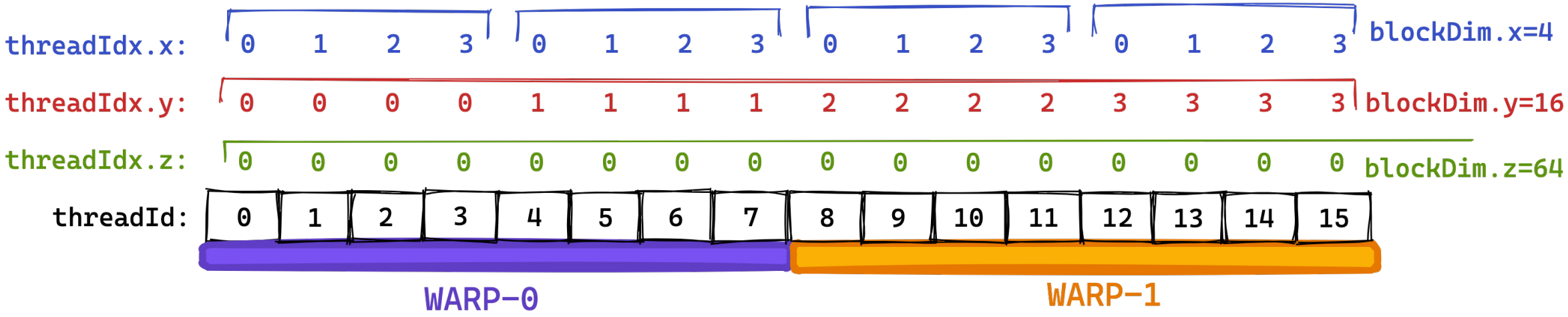
SiLU activation

warp_coalesced.py

```
cuda_source = """
__device__ float silu(float x) {
    return x / (1.0f + expf(-x));
}

__global__ void silu_uncoalesced_kernel(const float* input, float
    int col = blockIdx.y * blockDim.y + threadIdx.y;
    int row = blockIdx.x * blockDim.x + threadIdx.x;
    if (row < rows && col < cols) {
        int idx = row * cols + col;
        output[idx] = silu(input[idx]);
    }
}

torch::Tensor silu_uncoalesced(torch::Tensor input) {
    const int rows = input.size(0);
    const int cols = input.size(1);
    auto output = torch::empty_like(input);

    dim3 threads(32, 32);
    dim3 blocks((rows + threads.x - 1) / threads.x,
                (cols + threads.y - 1) / threads.y);
    silu_uncoalesced_kernel<<<blocks, threads>>>(
        input.data_ptr<float>(), output.data_ptr<float>(), rows,
    return output;
}
```

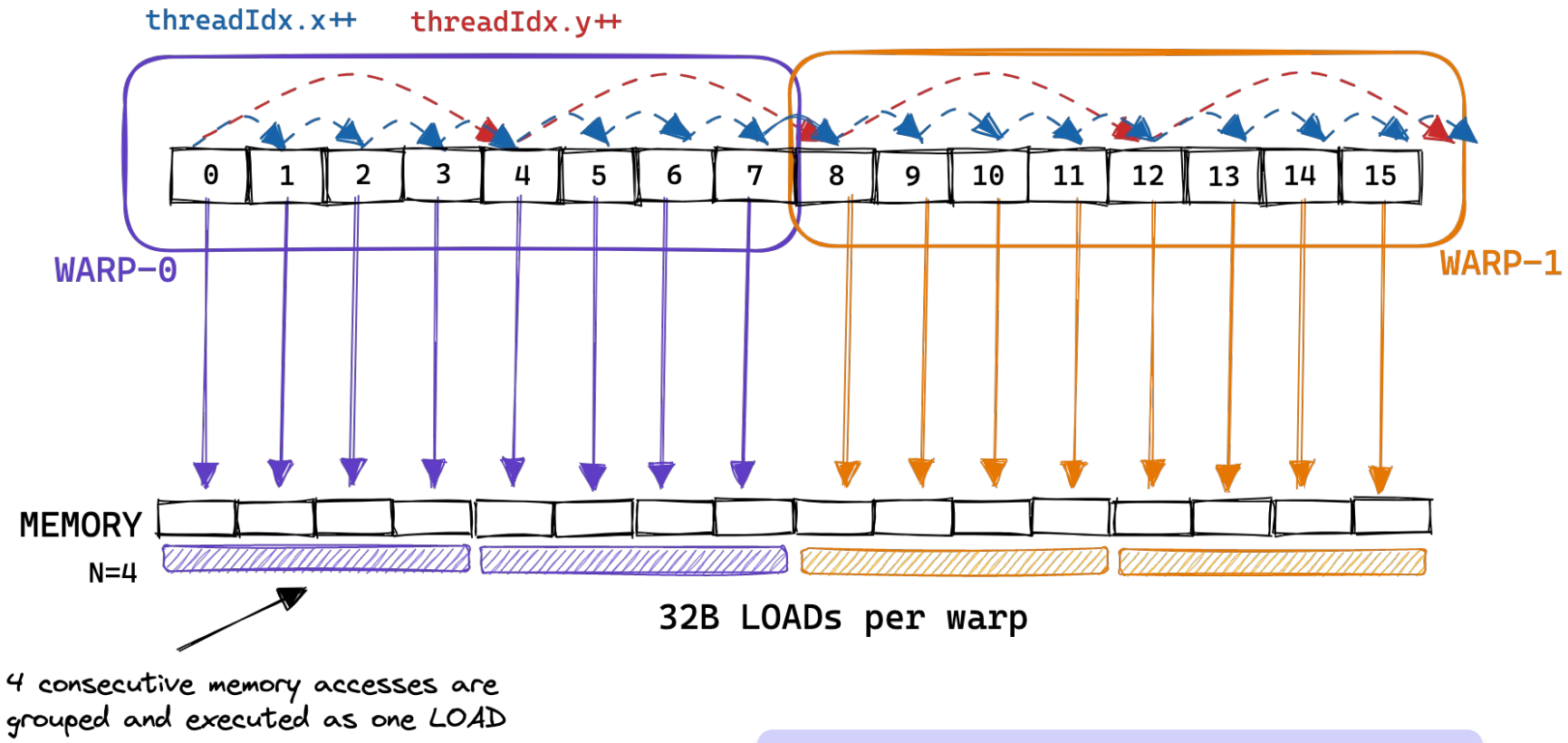


Warp: 线程束/线程簇



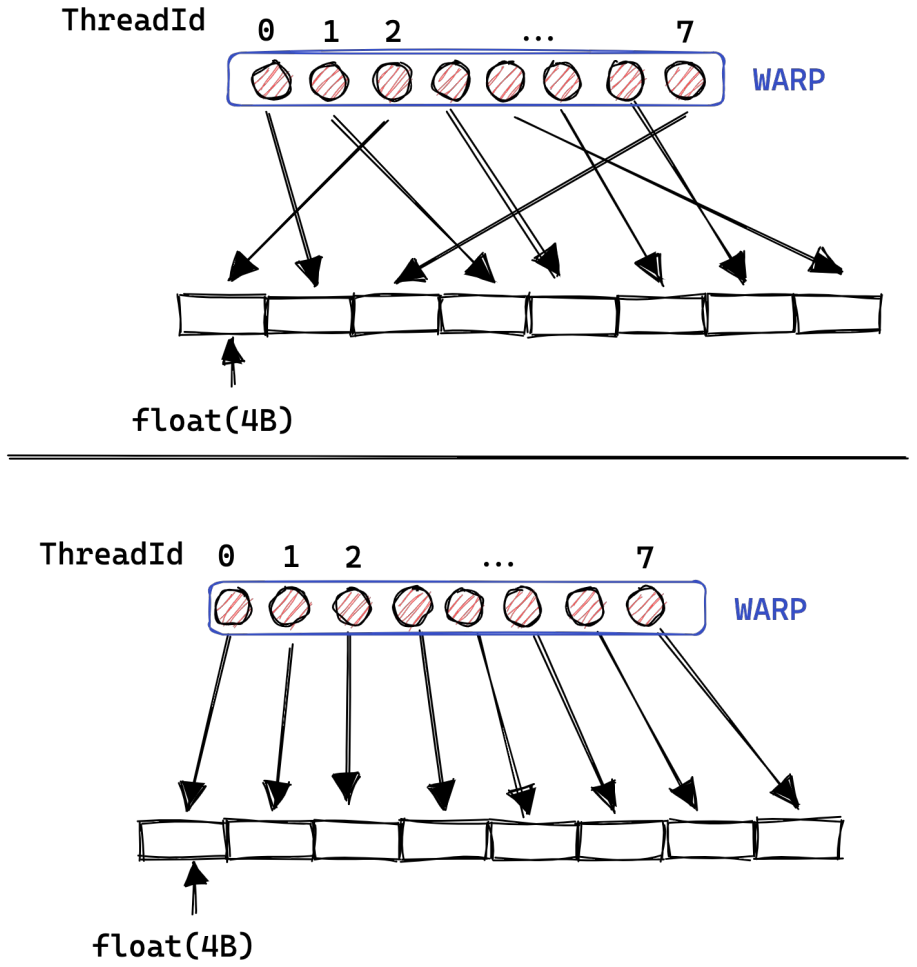
Warp=32

Warp: 线程束/线程簇



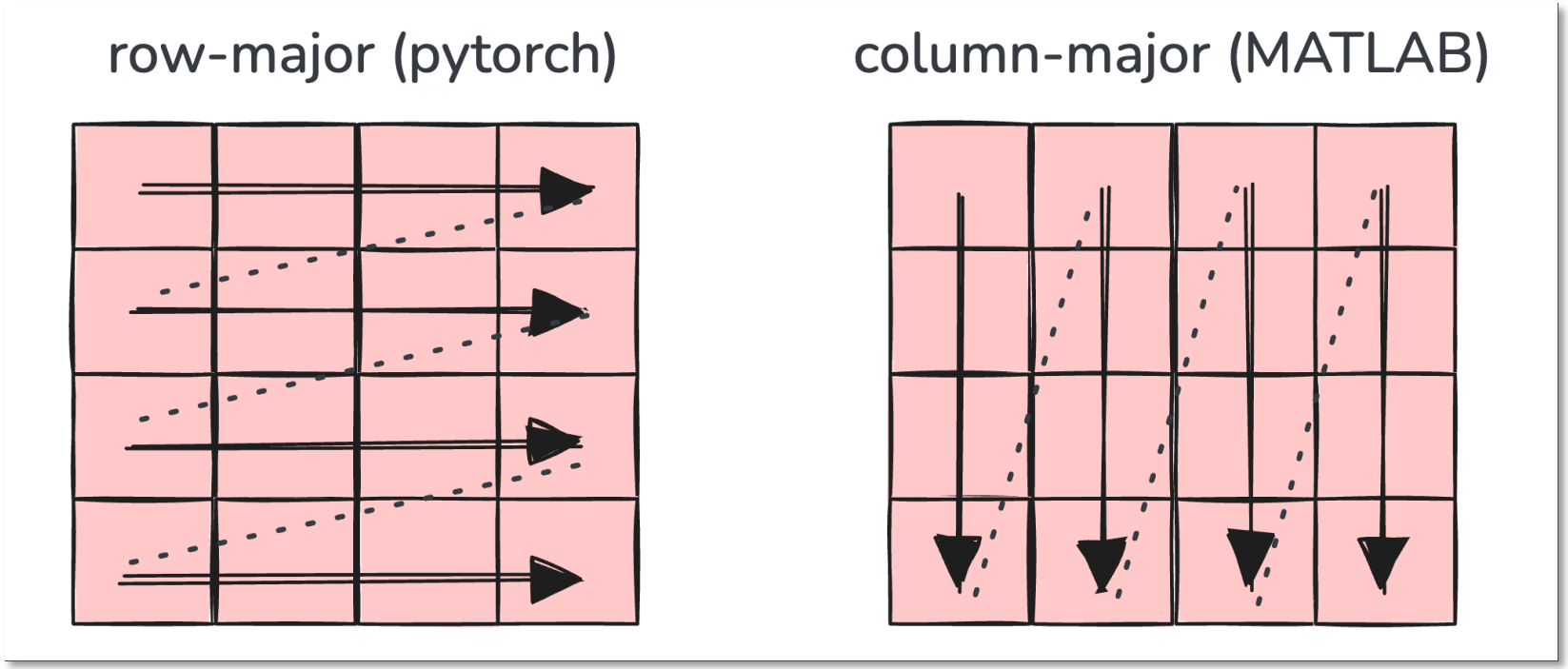
A&H支持32B/64B/128B

Warp: 线程束/线程簇

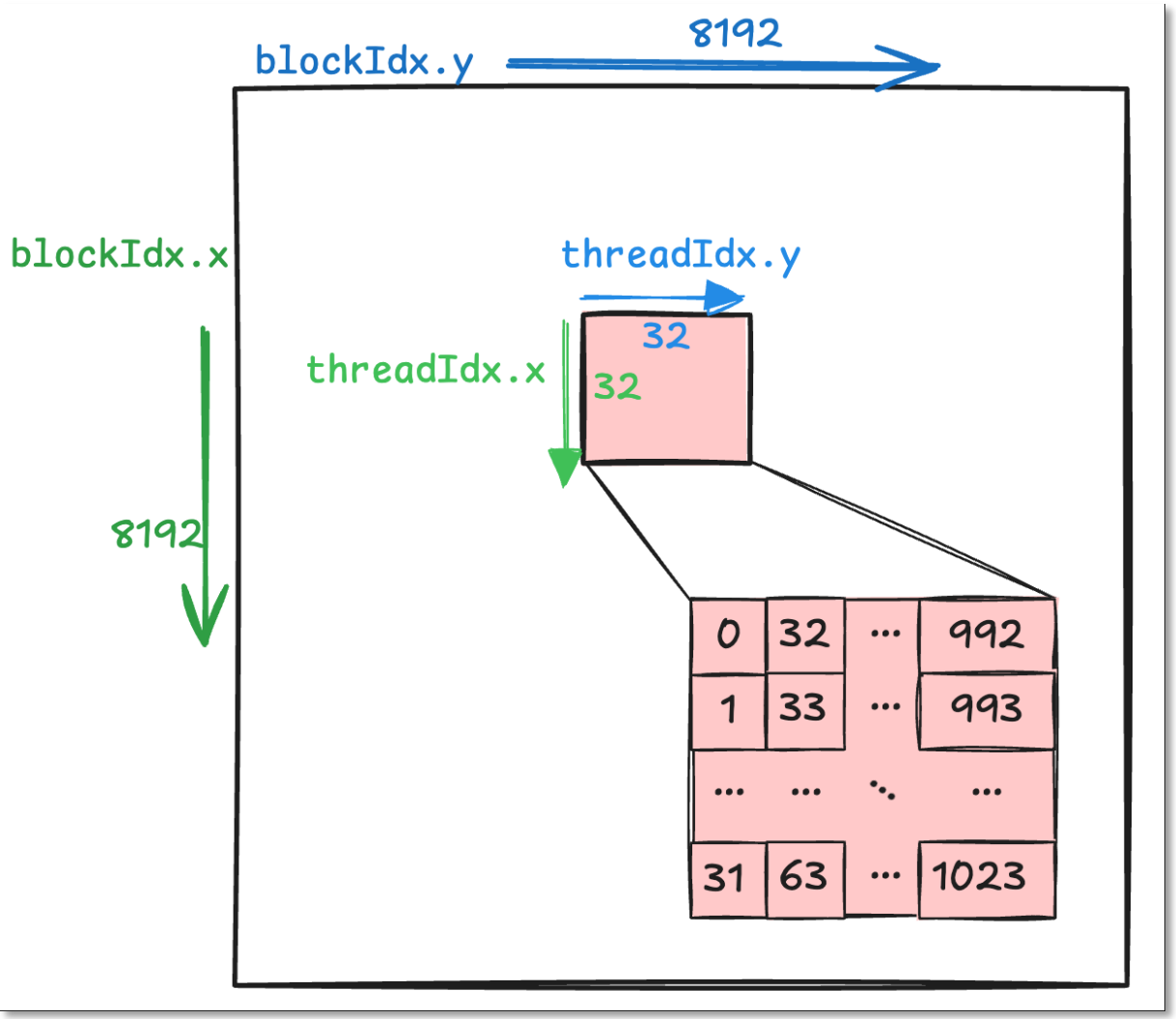


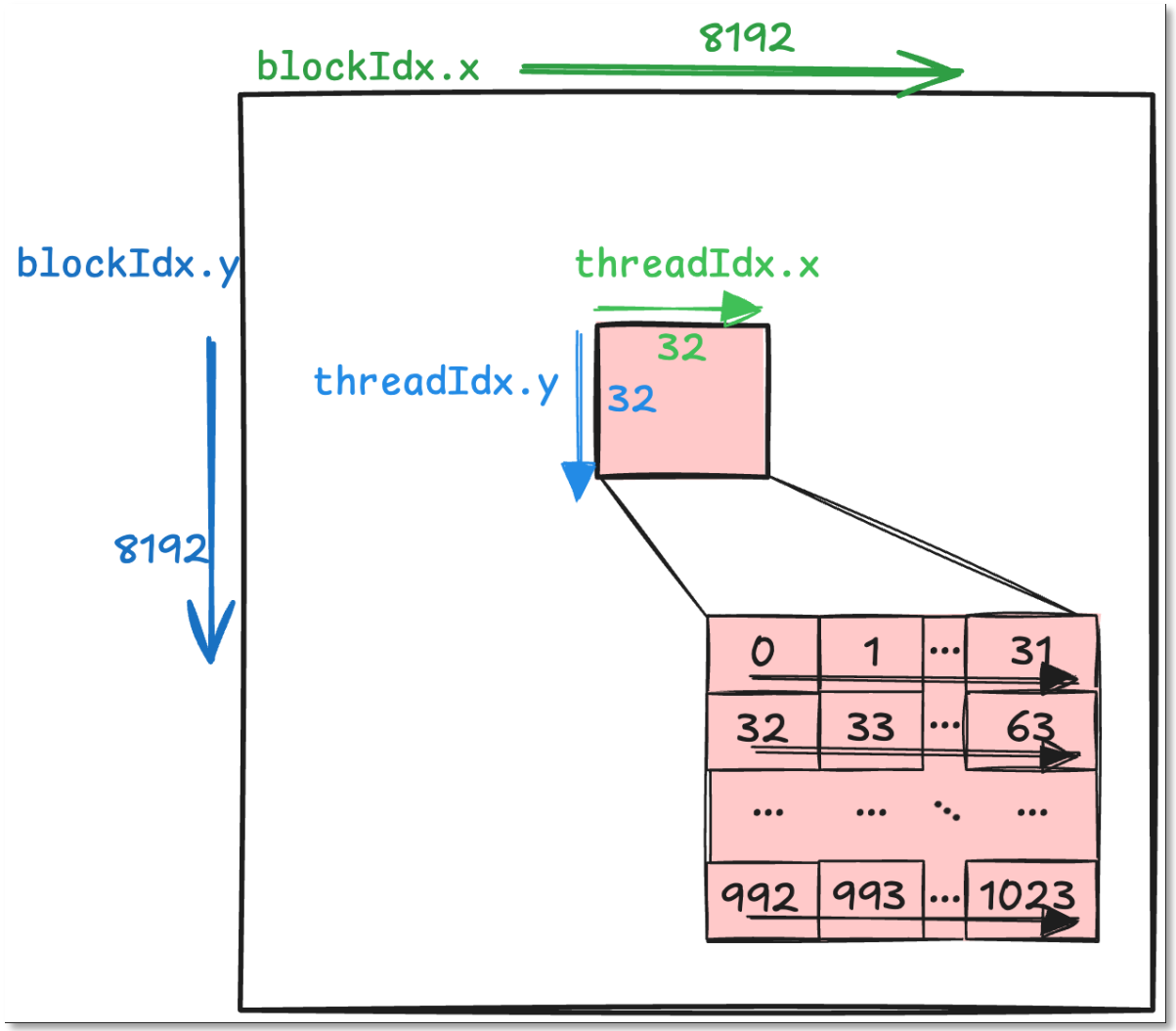
Both of these access patterns can be coalesced!

Contiguous



访存合并：Coalesced





访存合并：Coalesced

warp_coalesced.py

```
__global__ void silu_coalesced_kernel(const float* input, float* output, int rows, int cols) {
+    int col = blockIdx.x * blockDim.x + threadIdx.x;
+    int row = blockIdx.y * blockDim.y + threadIdx.y;
    if (row < rows && col < cols) {
        int idx = row * cols + col;
        output[idx] = silu(input[idx]);
    }
}

torch::Tensor silu_coalesced(torch::Tensor input) {
    const int rows = input.size(0);
    const int cols = input.size(1);
    auto output = torch::empty_like(input);

    dim3 threads(32, 32);
+    dim3 blocks((cols + threads.x - 1) / threads.x,
+                (rows + threads.y - 1) / threads.y);
    silu_coalesced_kernel<<<blocks, threads>>>(
        input.data_ptr<float>(), output.data_ptr<float>(), rows, cols);
    return output;
}
```

访存合并：Coalesced

```
Loading extension module moss_op...
-------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
                                                   Name    Self CPU %     Self CPU   CPU total %     CPU total  CPU time avg     Self CUDA  Self CUDA %    CUDA total  CUDA time avg
-------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
silu_uncoalesced_kernel(float const*, float*, int, i...       0.00%       0.000us       0.00%       0.000us       0.000us       7.175ms        81.10%       7.175ms       2.392ms
silu_coalesced_kernel(float const*, float*, int, int...       0.00%       0.000us       0.00%       0.000us       0.000us       1.067ms        12.06%       1.067ms     533.326us
void at::native::vectorized_elementwise_kernel<4, at...       0.00%       0.000us       0.00%       0.000us       0.000us     605.246us         6.84%     605.246us     302.623us
                                         cudaLaunchKernel       1.15%      69.085us       1.15%      69.085us      11.514us       0.000us         0.00%       0.000us       0.000us
                                    cudaDeviceSynchronize      98.85%       5.954ms      98.85%       5.954ms       5.954ms       0.000us         0.00%       0.000us       0.000us
-------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 6.024ms
Self CUDA time total: 8.847ms
```

向量化访存: Vectorized

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

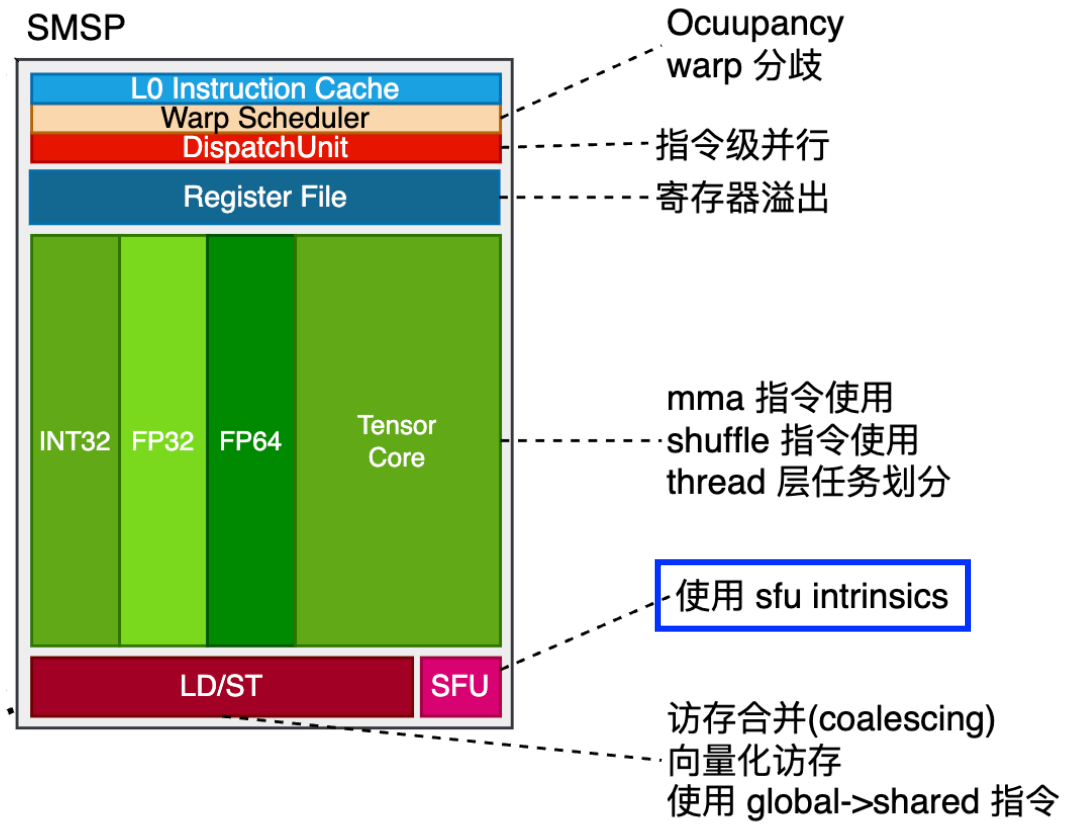
vectorized.py

```cpp
__global__ void silu_coalesced_float4_kernel(const float* input, float* output, int rows, int cols) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col4 = blockIdx.x * blockDim.x + threadIdx.x;
    int col = col4 << 2;

    if (row < rows && col + 3 < cols) {
        const float4* input_v4 = reinterpret_cast<const float4*>(input);
        float4 val = input_v4[row * (cols >> 2) + col4];

        val.x = silu(val.x);
        val.y = silu(val.y);
        val.z = silu(val.z);
        val.w = silu(val.w);

        float4* output_v4 = reinterpret_cast<float4*>(output);
        output_v4[row * (cols >> 2) + col4] = val;
    }
}

torch::Tensor silu_coalesced_float4(torch::Tensor input) {
    TORCH_CHECK(input.size(1) % 4 == 0, "Column size must be divisible by 4 for float4");

    int rows = input.size(0);
    int cols = input.size(1);

    auto output = torch::empty_like(input);

    dim3 threads(32, 32);
    dim3 blocks((cols / 4 + threads.x - 1) / threads.x, (rows + threads.y - 1) / threads.y);
    silu_coalesced_float4_kernel<<<blocks, threads>>>(
        input.data_ptr<float>(), output.data_ptr<float>(), rows, cols);
    return output;
}
```

向量化访存: Vectorized

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time av
silu_coalesced_float4_kernel(float const*, float*, i...	0.00%	0.000us	0.00%	0.000us	0.000us	638.077us	51.25%	638.077us	319.039us
void at::native::vectorized_elementwise_kernel<4, at...	0.00%	0.000us	0.00%	0.000us	0.000us	607.037us	48.75%	607.037us	303.519us
cudaLaunchKernel	4.70%	45.861us	4.70%	45.861us	11.465us	0.000us	0.00%	0.000us	0.000us
cudaDeviceSynchronize	95.30%	929.920us	95.30%	929.920us	929.920us	0.000us	0.00%	0.000us	0.000us

SFU Functions



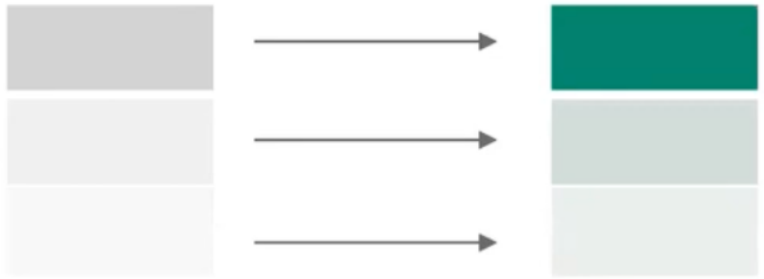
函数	描述
`__sinf(x)`	快速计算 `sin(x)`
`__cosf(x)`	快速计算 `cos(x)`
`__expf(x)`	快速计算 `exp(x)`
`__logf(x)`	快速计算 `log(x)`
`__powf(x, y)`	快速计算 `x^y`
`__sqrtf(x)`	快速计算 `sqrt(x)`
`__fdividef(a, b)`	快速除法

SFU Functions

sfu.py

```
- 1  __device__ float silu(float x) {
- 2      return x / (1.0f + expf(-x));
- 3  }
  4
+ 5  __device__ float silu_sfu(float x) {
+ 6      return x / (1.0f + __expf(-x));
+ 7  }
```

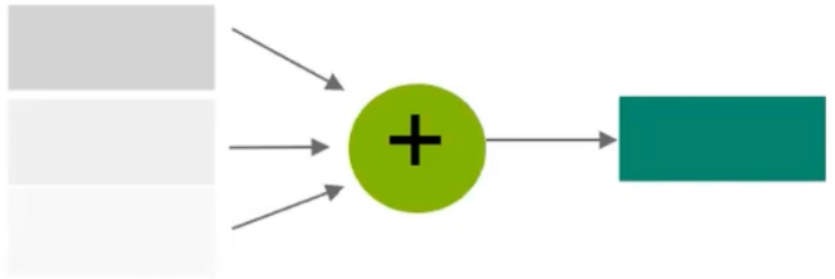
Transformation vs Reduction



Transformation:

e.g. c[i] = a[i] + 10;

Thread strategy: one thread per output point

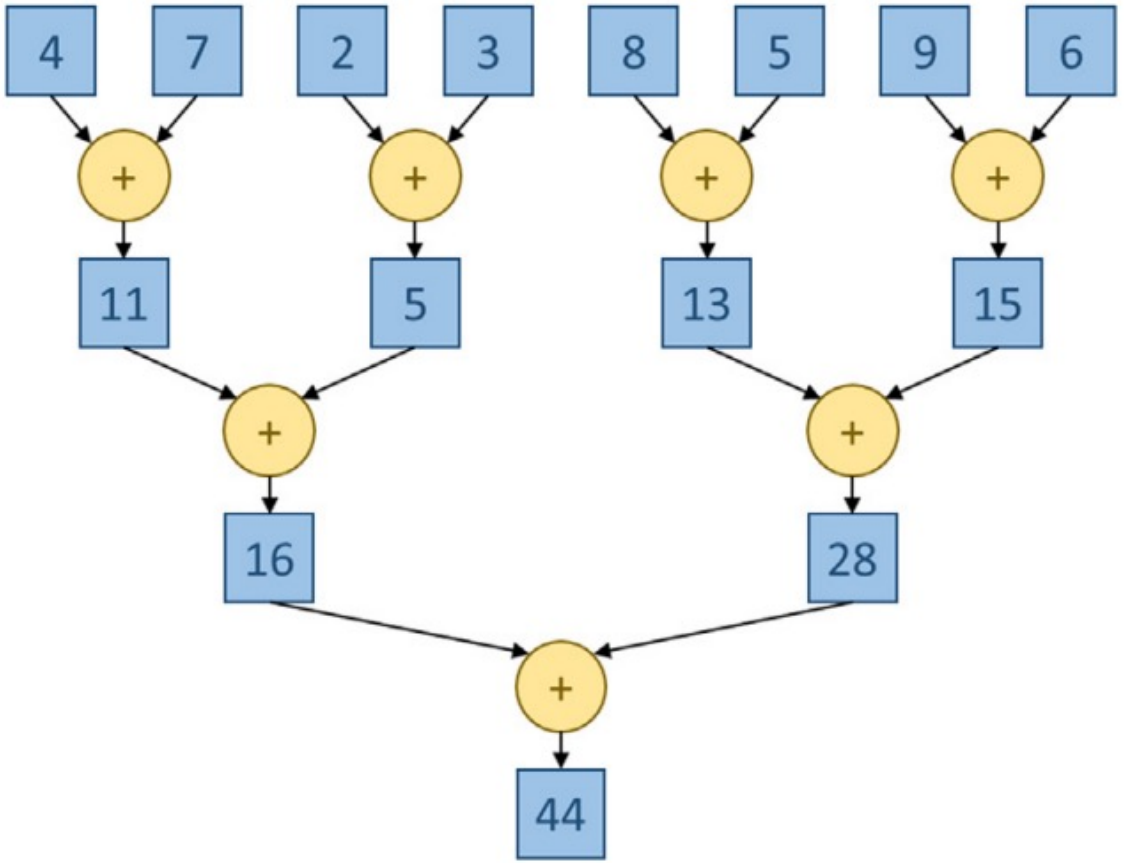


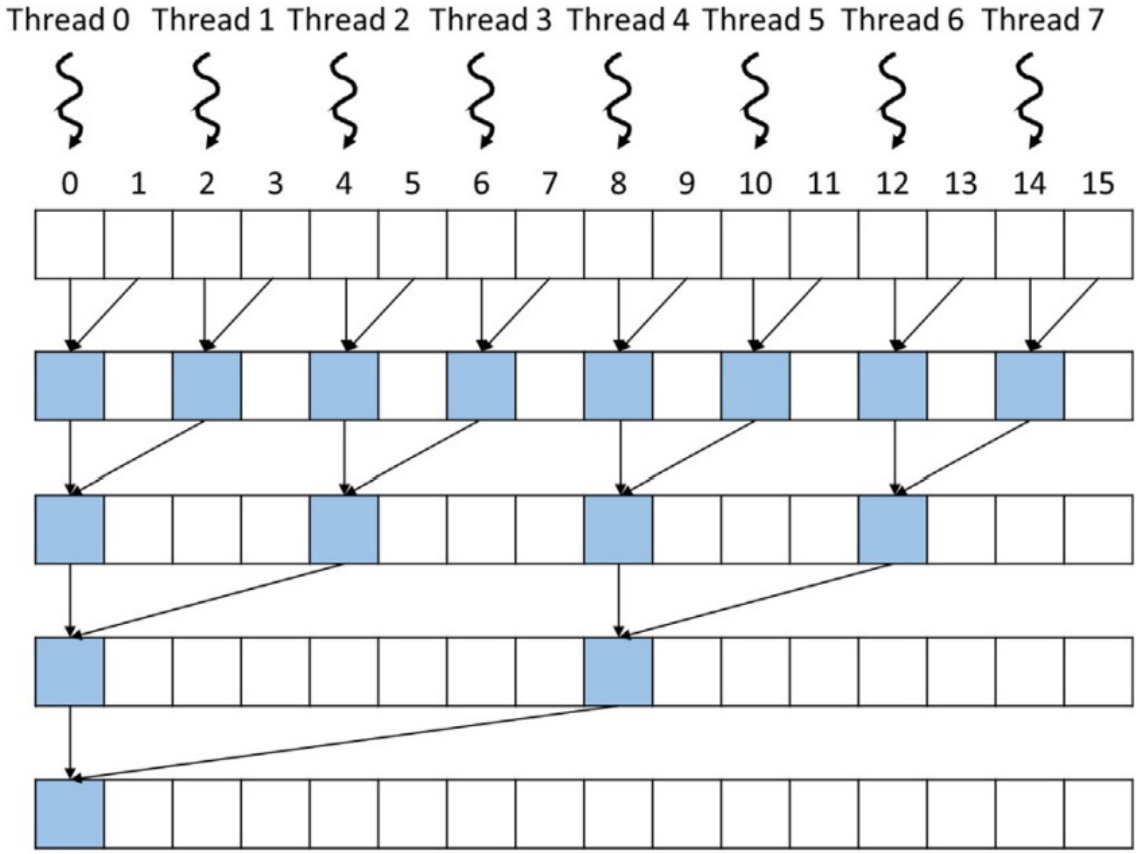
Reduction:

e.g. *c = Σ a[i]

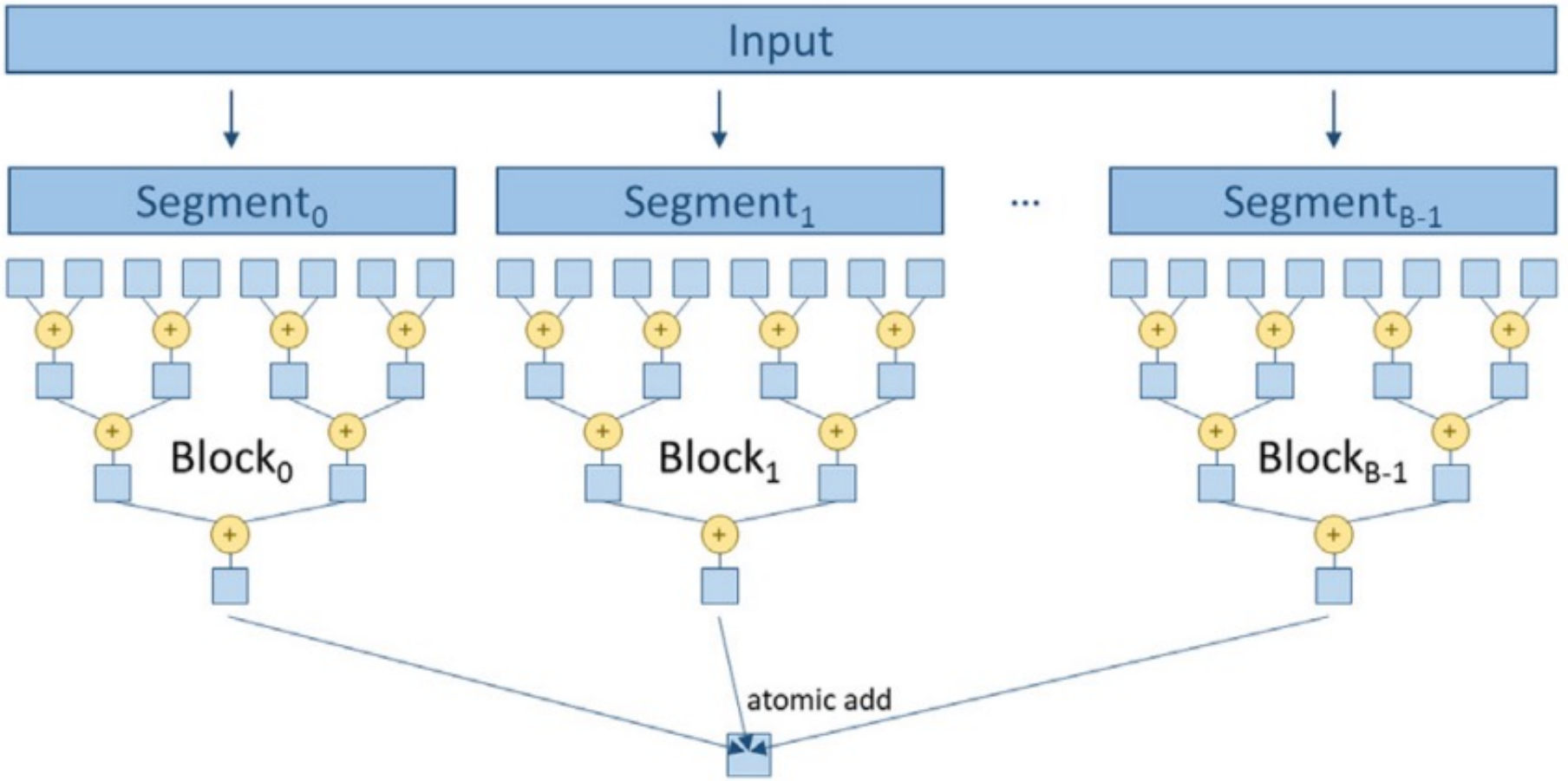
Thread strategy: ??

Reduction Tree





Reduction Tree



naive_reduce.py

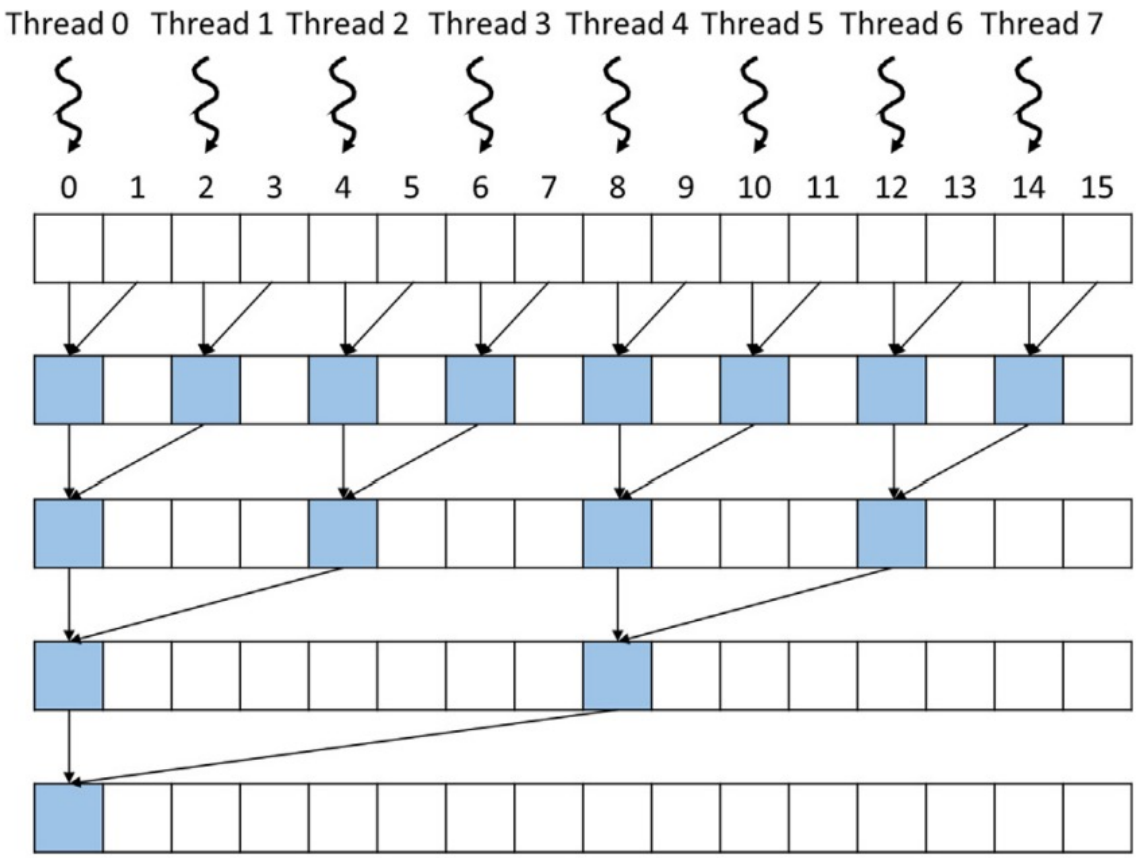
```
cuda_source = """
__global__ void reduce_neighbor_atomic_kernel(const float* input, float* output, int N) {
    __shared__ float sdata[128];

    uint tid = threadIdx.x;
    uint i = blockIdx.x * blockDim.x * 2 + tid;

    float val = 0.0f;
    if (i < N) val += input[i];
    if (i + blockDim.x < N) val += input[i + blockDim.x];
    sdata[tid] = val;

    __syncthreads();

    for (int stride = 1; stride < blockDim.x; stride <<= 1) {
        if (tid % (stride<<1) == 0 && tid + stride < blockDim.x)
            sdata[tid] += sdata[tid + stride];
        __syncthreads();
    }

    if (tid == 0) {
        atomicAdd(output, sdata[0]);
    }
}

torch::Tensor reduce_neighbor_atomic(torch::Tensor input) {
    int N = input.size(0);
    int threads = 128;
    int blocks = (N + threads * 2 - 1) / (threads * 2);

    auto output = torch::zeros({}, input.options());

    reduce_neighbor_atomic_kernel<<<blocks, threads>>>(
        input.data_ptr<float>(), output.data_ptr<float>(), N
    );

    return output;
}
"""
```

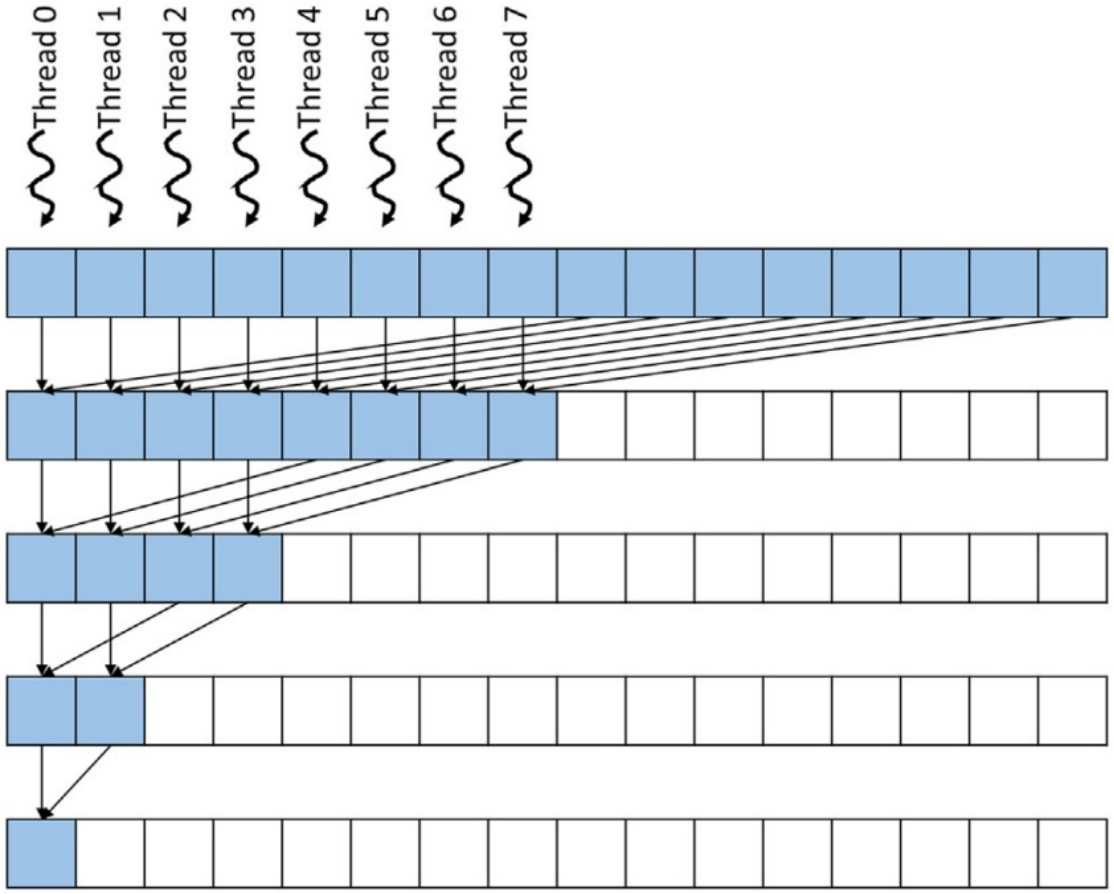
naive_reduce.py

```
cpp_source = "torch::Tensor reduce_neighbor_atomic(torch::Tensor input);"

module = load_inline(
    name="moss_op",
    cpp_sources=cpp_source,
    cuda_sources=cuda_source,
    functions=["reduce_neighbor_atomic"],
    verbose=False,
)

N = 1 << 20
x = torch.randn(N, device="cuda")

with profile(
    activities=[ProfilerActivity.CUDA],
    schedule=torch.profiler.schedule(wait=1, warmup=1, active=2),
    on_trace_ready=torch.profiler.tensorboard_trace_handler("./log/reduce_atomic_prof"),
    record_shapes=False,
    with_stack=False,
) as prof:
    for i in range(4):
        with record_function("cuda_reduce_atomic"), torch.no_grad():
            out1 = module.reduce_neighbor_atomic(x)
        with record_function("torch_sum"), torch.no_grad():
            out0 = torch.sum(x)
        prof.step()

assert torch.allclose(out0, out1)
print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

Reduction Tree

- A lot of threads will be inactive :(





Reduction

reorder_reduce.py

```
cuda_source = """
__global__ void reduce_reorder_atomic_kernel(const float* input, float* output, int N) {
    __shared__ float sdata[1024];

    uint tid = threadIdx.x;
    uint i = blockIdx.x * blockDim.x * 2 + tid;

    float val = 0.0f;
    if (i < N) val += input[i];
    if (i + blockDim.x < N) val += input[i + blockDim.x];
    sdata[tid] = val;

    __syncthreads();

+   for (int s = blockDim.x / 2; s > 0; s >>= 1) {
+       if (tid < s)
+           sdata[tid] += sdata[tid + s];
        __syncthreads();
    }

    if (tid == 0) {
        atomicAdd(output, sdata[0]);
    }
}

torch::Tensor reduce_reorder_atomic(torch::Tensor input) {
    int N = input.size(0);
    int threads = 1024;
    int blocks = (N + threads * 2 - 1) / (threads * 2);

    auto output = torch::zeros({}, input.options());

    reduce_reorder_atomic_kernel<<<blocks, threads>>>(
        input.data_ptr<float>(), output.data_ptr<float>(), N
    );

    return output;
}
"""
```

Reduction

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg
reduce_reorder_atomic_kernel(float const*, float*, i...	0.00%	0.000us	0.00%	0.000us	0.000us	21.536us	47.09%	21.536us	10.768us
void at::native::reduce_kernel<512, 1, at::native::R...	0.00%	0.000us	0.00%	0.000us	0.000us	18.368us	40.17%	18.368us	9.184us
void at::native::vectorized_elementwise_kernel<4, at...	0.00%	0.000us	0.00%	0.000us	0.000us	3.265us	7.14%	3.265us	1.632us
Memset (Device)	0.00%	0.000us	0.00%	0.000us	0.000us	2.560us	5.60%	2.560us	1.280us
cudaLaunchKernel	71.60%	83.565us	71.60%	83.565us	13.928us	0.000us	0.00%	0.000us	0.000us
cudaMemsetAsync	18.79%	21.930us	18.79%	21.930us	10.965us	0.000us	0.00%	0.000us	0.000us
cudaDeviceSynchronize	9.61%	11.216us	9.61%	11.216us	11.216us	0.000us	0.00%	0.000us	0.000us

Reduction

作业二：+ Vectorized

reorder_reduce.py

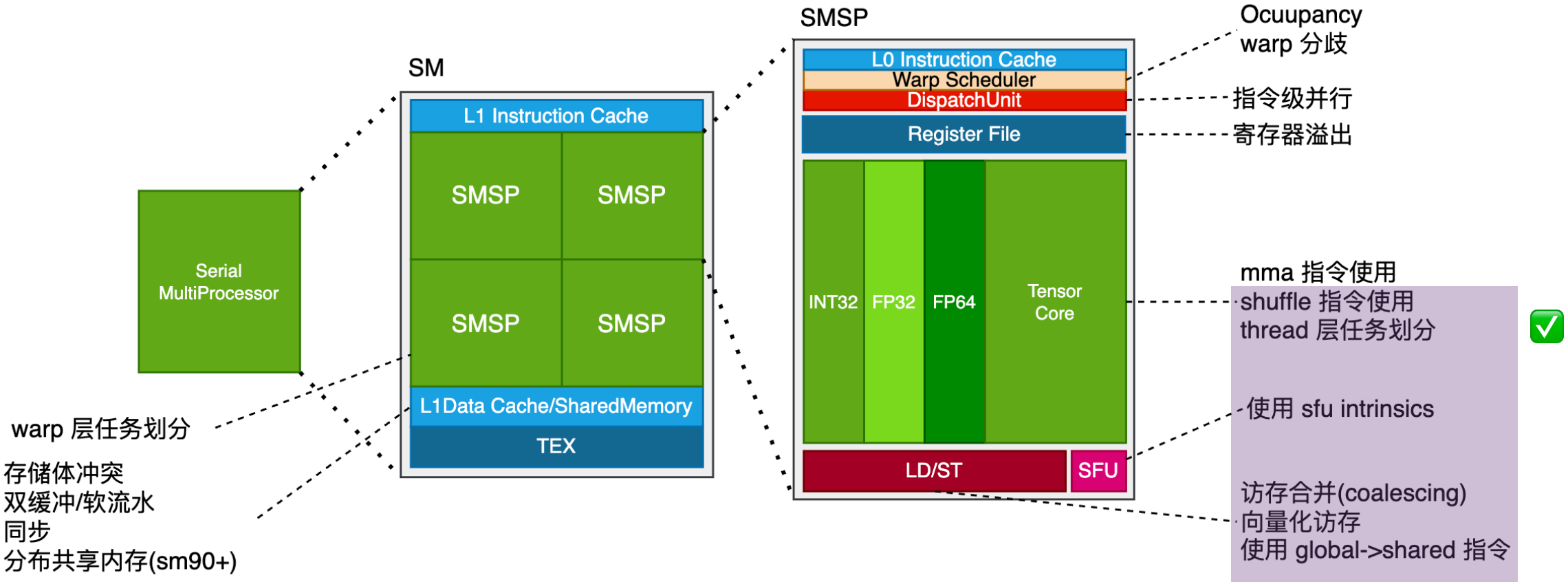
```
cuda_source = """
__global__ void reduce_reorder_atomic_kernel(const float* input, float* output, int N) {
    __shared__ float sdata[1024];

    uint tid = threadIdx.x;
    uint i = blockIdx.x * blockDim.x * 2 + tid;

    float val = 0.0f;
    if (i < N) val += input[i];
    if (i + blockDim.x < N) val += input[i + blockDim.x];
    sdata[tid] = val;

    __syncthreads();

+   for (int s = blockDim.x / 2; s > 0; s >>= 1) {
+       if (tid < s)
+           sdata[tid] += sdata[tid + s];
        __syncthreads();
    }

    if (tid == 0) {
        atomicAdd(output, sdata[0]);
    }
}

torch::Tensor reduce_reorder_atomic(torch::Tensor input) {
    int N = input.size(0);
    int threads = 1024;
    int blocks = (N + threads * 2 - 1) / (threads * 2);

    auto output = torch::zeros({}, input.options());

    reduce_reorder_atomic_kernel<<<blocks, threads>>>(
        input.data_ptr<float>(), output.data_ptr<float>(), N
    );

    return output;
}
"""
```

Shuffle指令

4_shuffle_reduce.py

```
cuda_source = """
__inline__ __device__ float warp_reduce_sum(float val) {
    for (int8_t offset = 16; offset > 0; offset >>= 1)
        val += __shfl_down_sync(0xffffffff, val, offset);
    return val;
}

__global__ void reduce_shuffle_atomic_kernel(const float* input, float* output, int N) {
    __shared__ float shared[32];

    float val = 0.0f;
    int idx = blockIdx.x * blockDim.x * 2 + threadIdx.x;

    if (idx < N) val += input[idx];
    if (idx + blockDim.x < N) val += input[idx + blockDim.x];

    val = warp_reduce_sum(val);

    if ((threadIdx.x & 31) == 0)
        shared[threadIdx.x >> 5] = val;

    __syncthreads();

    float block_sum = 0.0f;
    if (threadIdx.x < 32) {
        block_sum = shared[threadIdx.x];
        block_sum = warp_reduce_sum(block_sum);
    }

    if (threadIdx.x == 0)
        atomicAdd(output, block_sum);
}
```

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
INT32
FP32
FP64
Tensor Core
mma 指令使用
shuffle 指令使用
thread 层任务划分
Serial MultiProcessor
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

Research Proposal

- Efficient infrastructure via algorithm-hardware co-design
- Language-centric multimodal in-context learning

FlexMLA

Optimize + Test

SlimFFN

Optimize + Test

FoX

Reproduction

taoji@fudan.edu.cn
A5029, 江湾叉二

Q&A