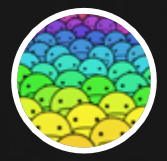
AI 芯片 – GPU 详解

Tensor Core 深度剖析



ZOMI

HUAWEI Ascend (www.hiascend.com)

Talk Overview

1. **硬件基础**
   - GPU 工作原理
   - GPU AI编程本质
2. **英伟达 GPU 架构**
   - GPU基础概念
   - 从 Fermi 到 Volta 架构
   - Turing 到 Hopper 架构
   - Tensor Code 和 NVLink 详解
3. **GPU 图形处理**
   - GPU 逻辑模块划分
   - 算法到 GPU 硬件
   - GPU 的软件栈
   - 图形流水线基础
   - 流水线不可编译单元
   - 光线跟踪流水线

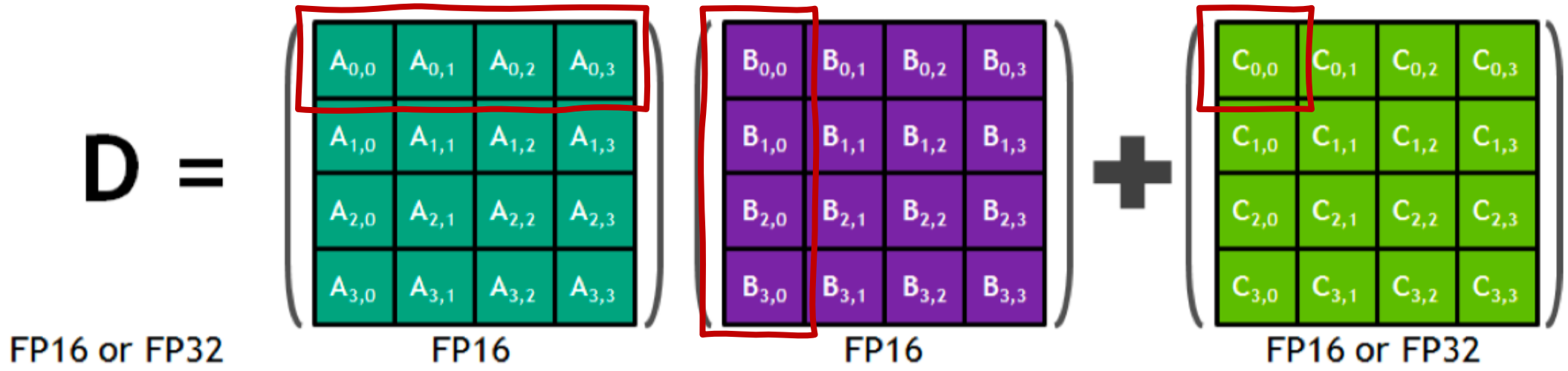
Talk Overview

I. 工作原理

- Tensor Core – Tensor Core 执行
- Instruction Pipeline – 指令流水
- CUDA Thread – CUDA 线程执行

Tensor Core
执行

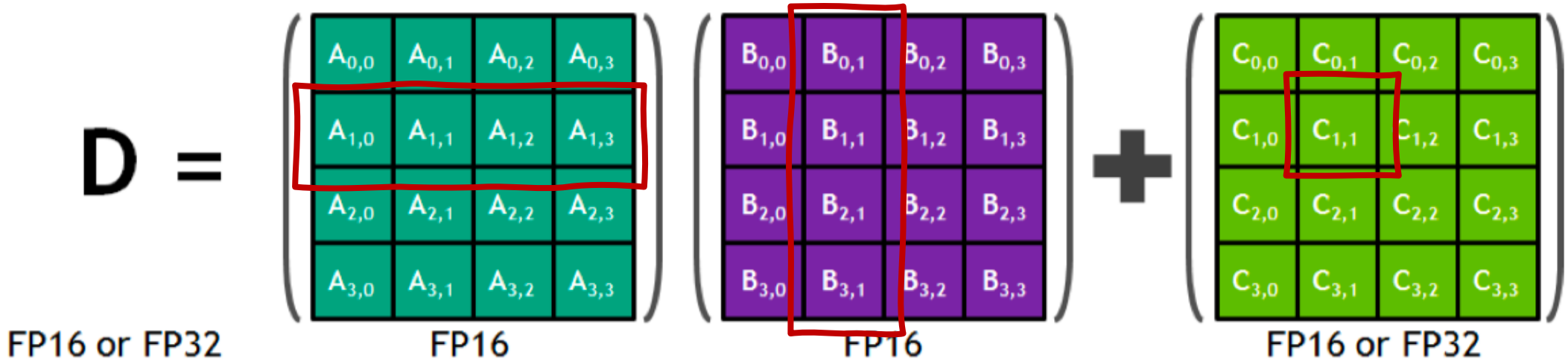
Tensor Core



- FMA: *D=A*B+C*

$$D_{0,0} = A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} + A_{0,2} * B_{2,0} + A_{0,3} * B_{3,0} + C_{0,0}$$

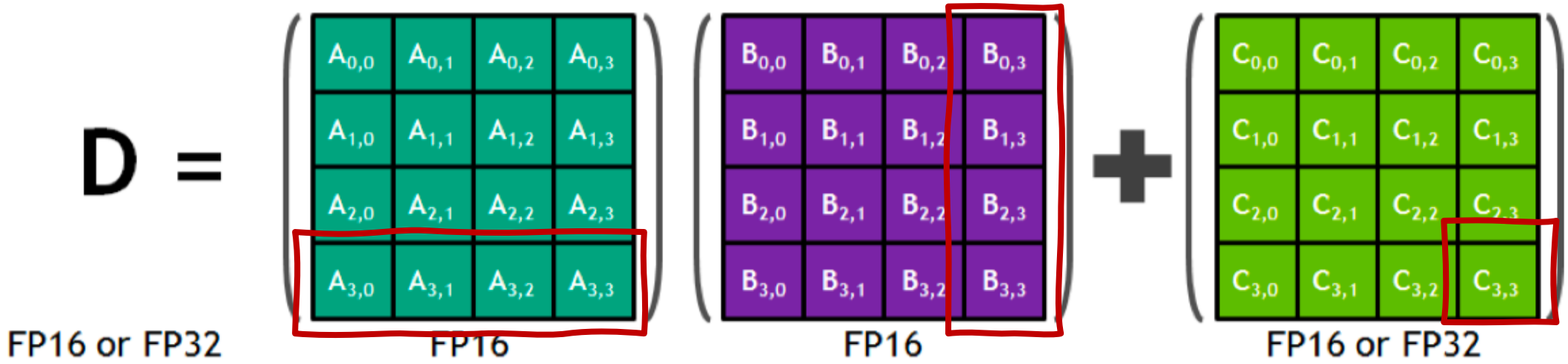
Tensor Core



$$D_{0,0} = A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} + A_{0,2} * B_{2,0} + A_{0,3} * B_{3,0} + C_{0,0}$$

$$D_{1,1} = A_{1,0} * B_{0,1} + A_{1,1} * B_{1,1} + A_{1,2} * B_{2,1} + A_{1,3} * B_{3,1} + C_{1,1}$$

Tensor Core



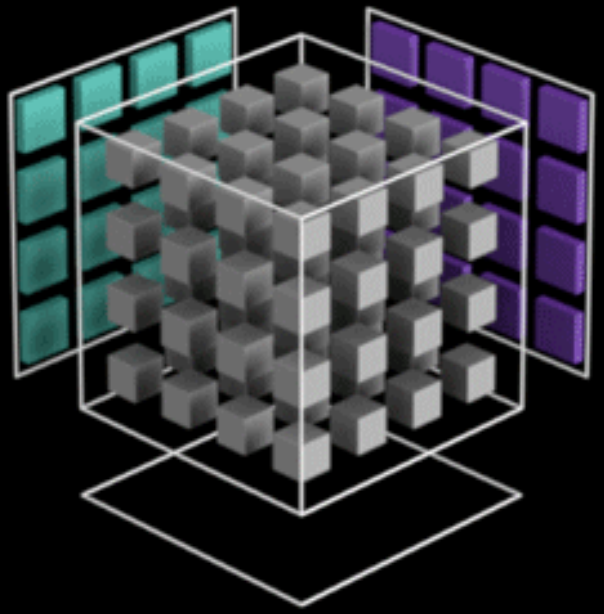
$$D_{0,0} = A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} + A_{0,2} * B_{2,0} + A_{0,3} * B_{3,0} + C_{0,0}$$

$$D_{1,1} = A_{1,0} * B_{0,1} + A_{1,1} * B_{1,1} + A_{1,2} * B_{2,1} + A_{1,3} * B_{3,1} + C_{1,1}$$

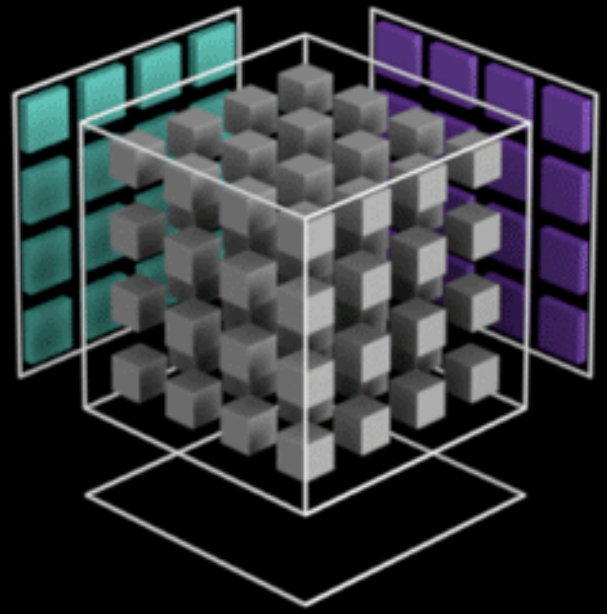
...

$$D_{3,3} = A_{3,0} * B_{0,3} + A_{3,1} * B_{1,3} + A_{3,2} * B_{2,3} + A_{3,3} * B_{3,3} + C_{3,3}$$

PASCAL

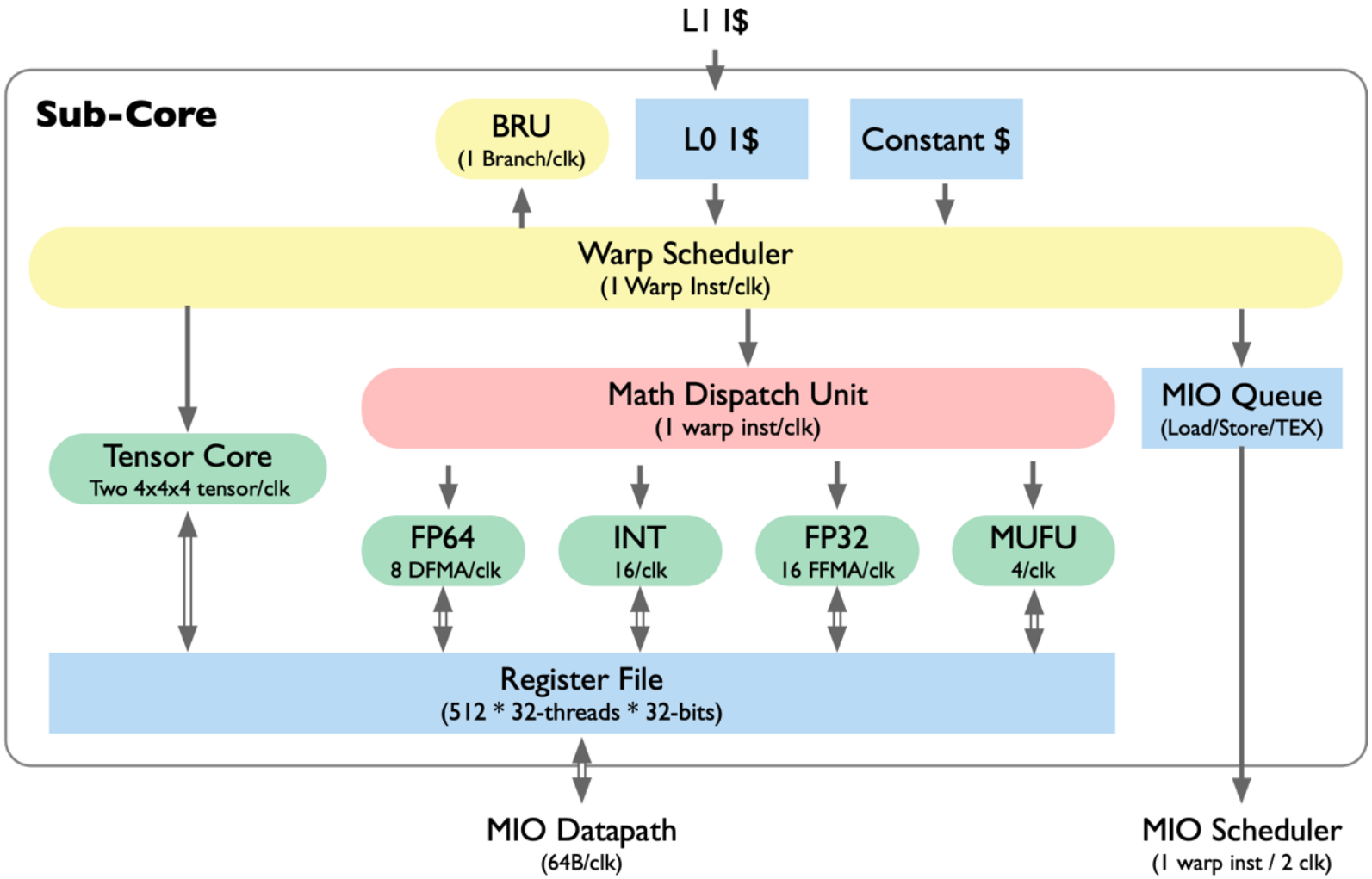


VOLTA TENSOR CORES



Tensor Core 单时钟周期内能执行 4×4×4=64 次 FMA（Fused-Multiply-Add，乘加计算）

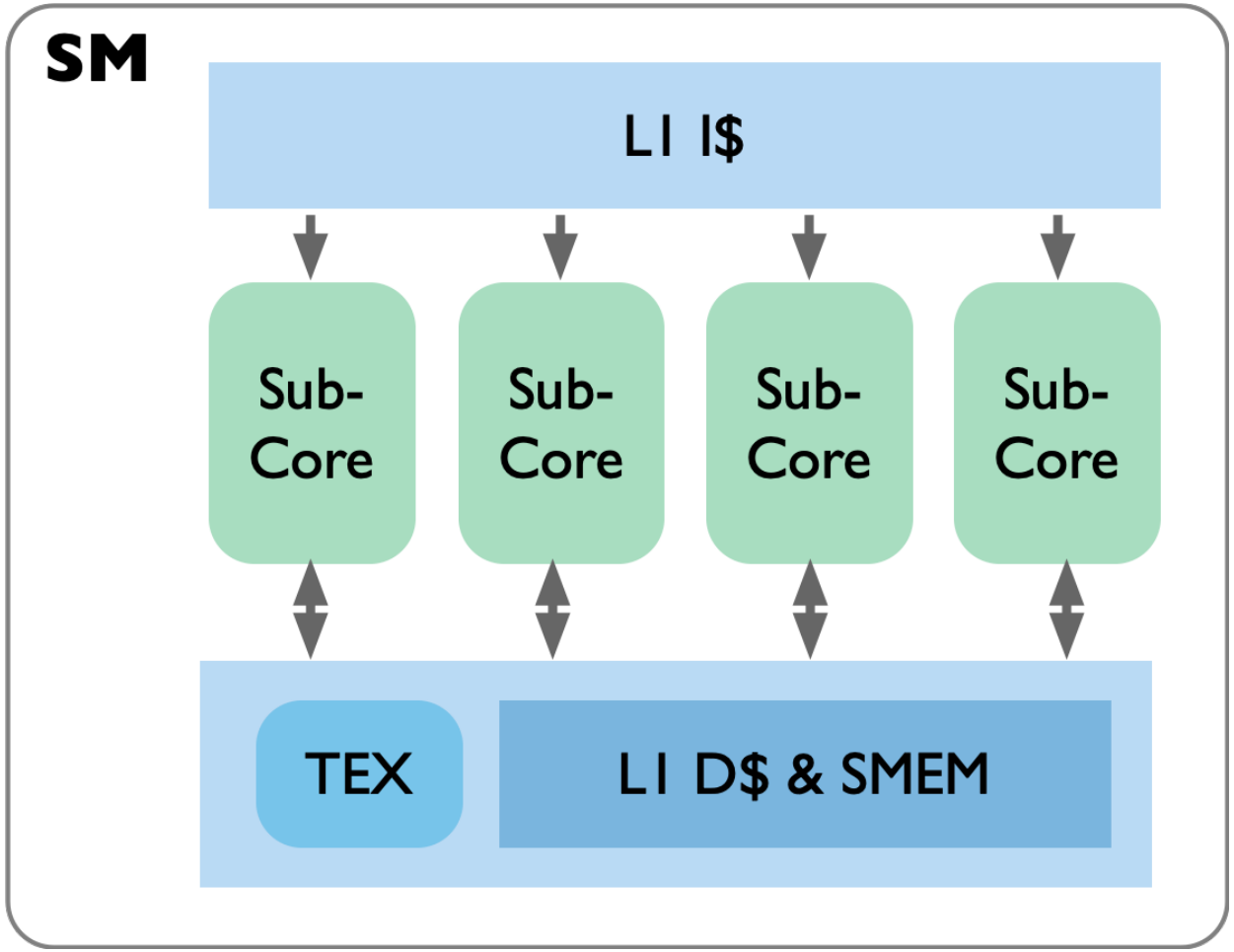
VOLTA Sub-Core 微架构



- **Warp Scheduler**
  - 1 Warp instr/clk
  - L0 I$, branch unit
- **Math Dispatch Unit**
  - Keeps 2+ Datapaths Busy
- **MIO Instruction Queue**
  - Hold for later Scheduling
- **Tensor Cores**
  - Two 4x4x4 matric

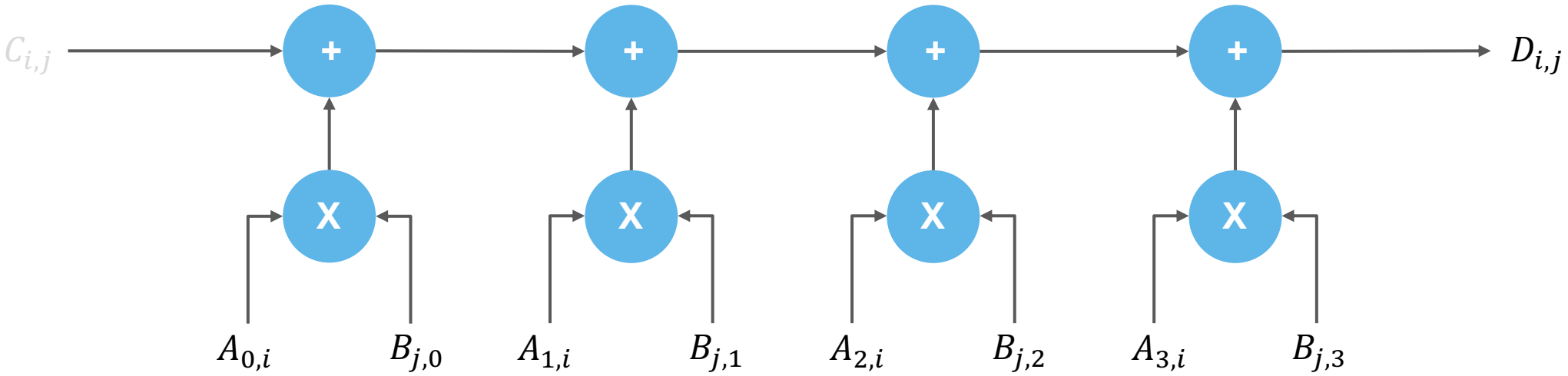
VOLTA SM


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex


SM
L1 I$
Sub-Core
Sub-Core
Sub-Core
Sub-Core
TEX
L1 D$ & SMEM

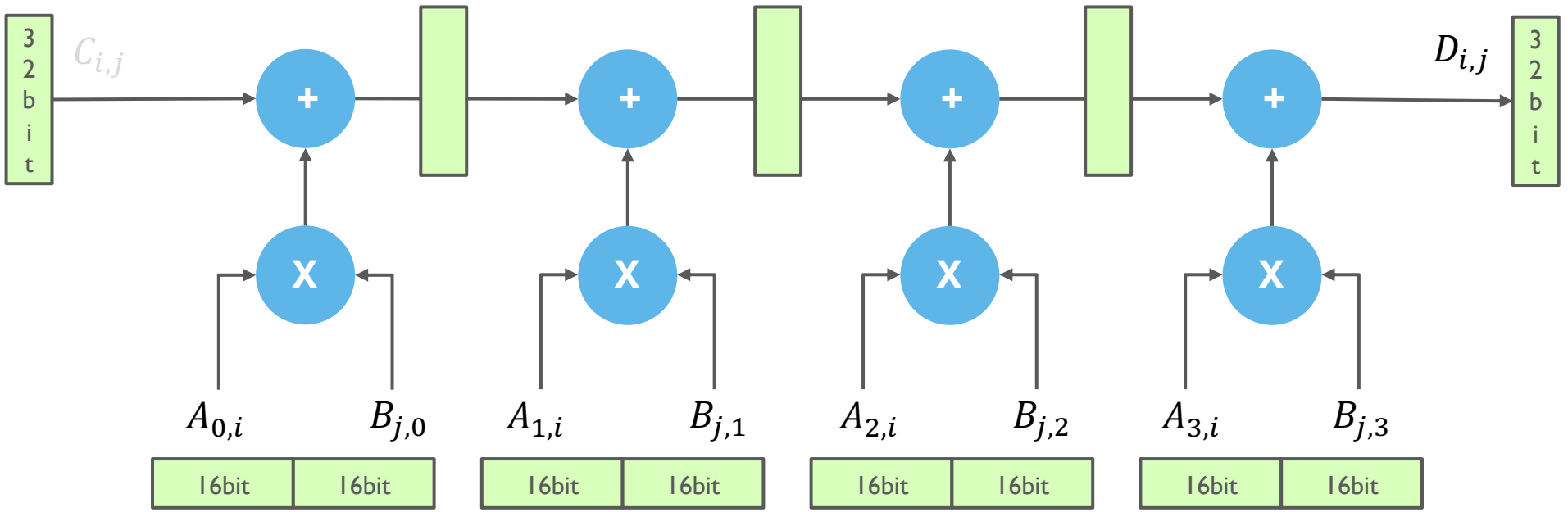
Tensor Core
指令流水

High Level Block Diagram



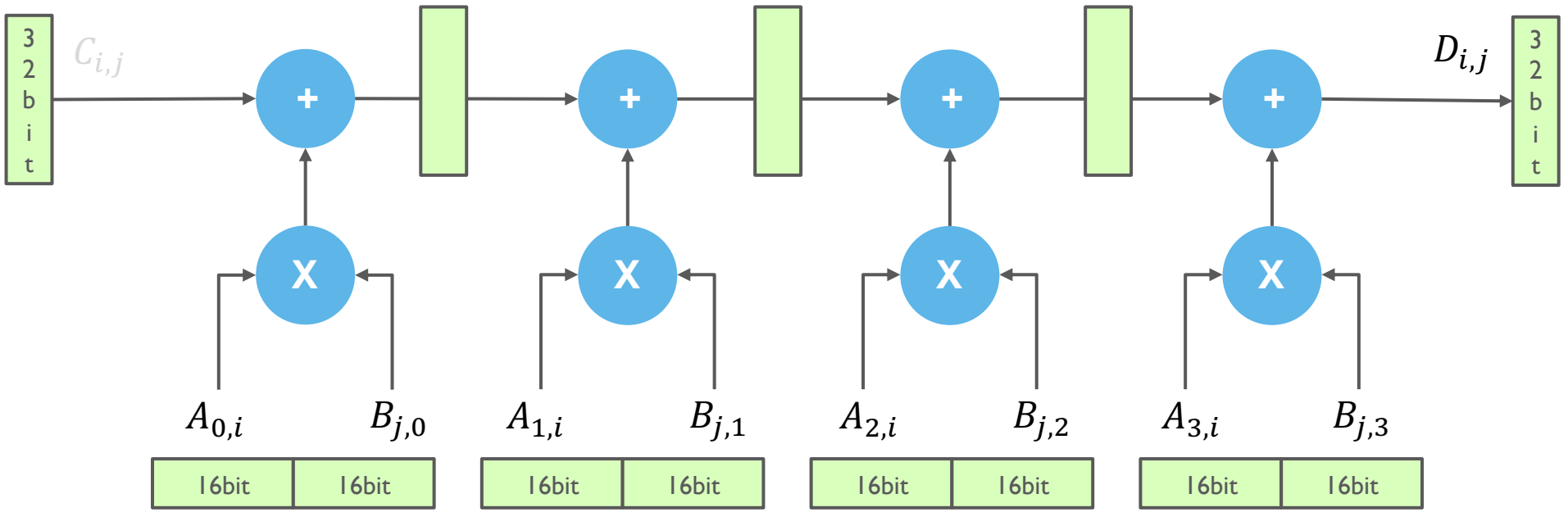
$$D_{0,0} = A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} + A_{0,2} * B_{2,0} + A_{0,3} * B_{3,0} + C_{0,0}$$

High Level Block Diagram



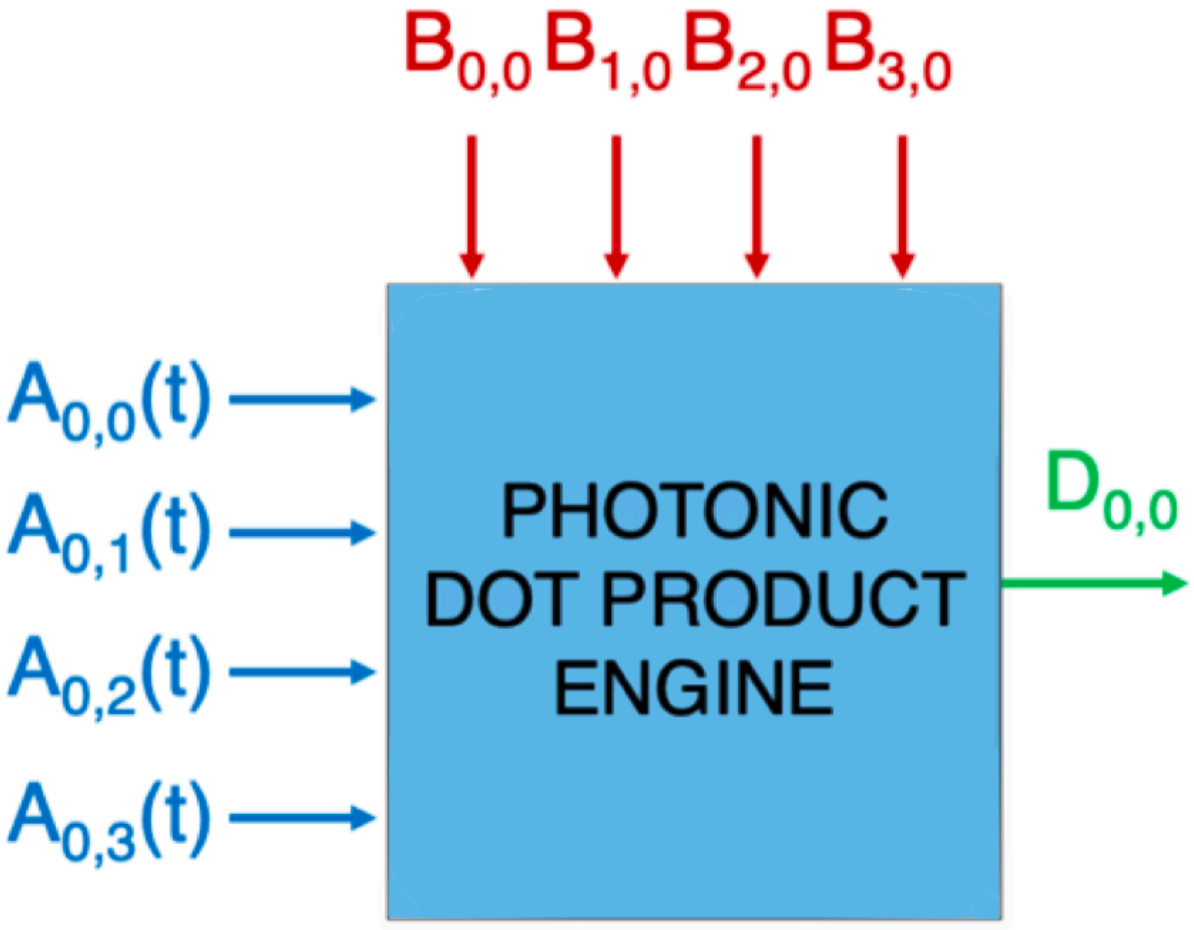
$$D_{0,0} = A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} + A_{0,2} * B_{2,0} + A_{0,3} * B_{3,0} + C_{0,0}$$

High Level Block Diagram

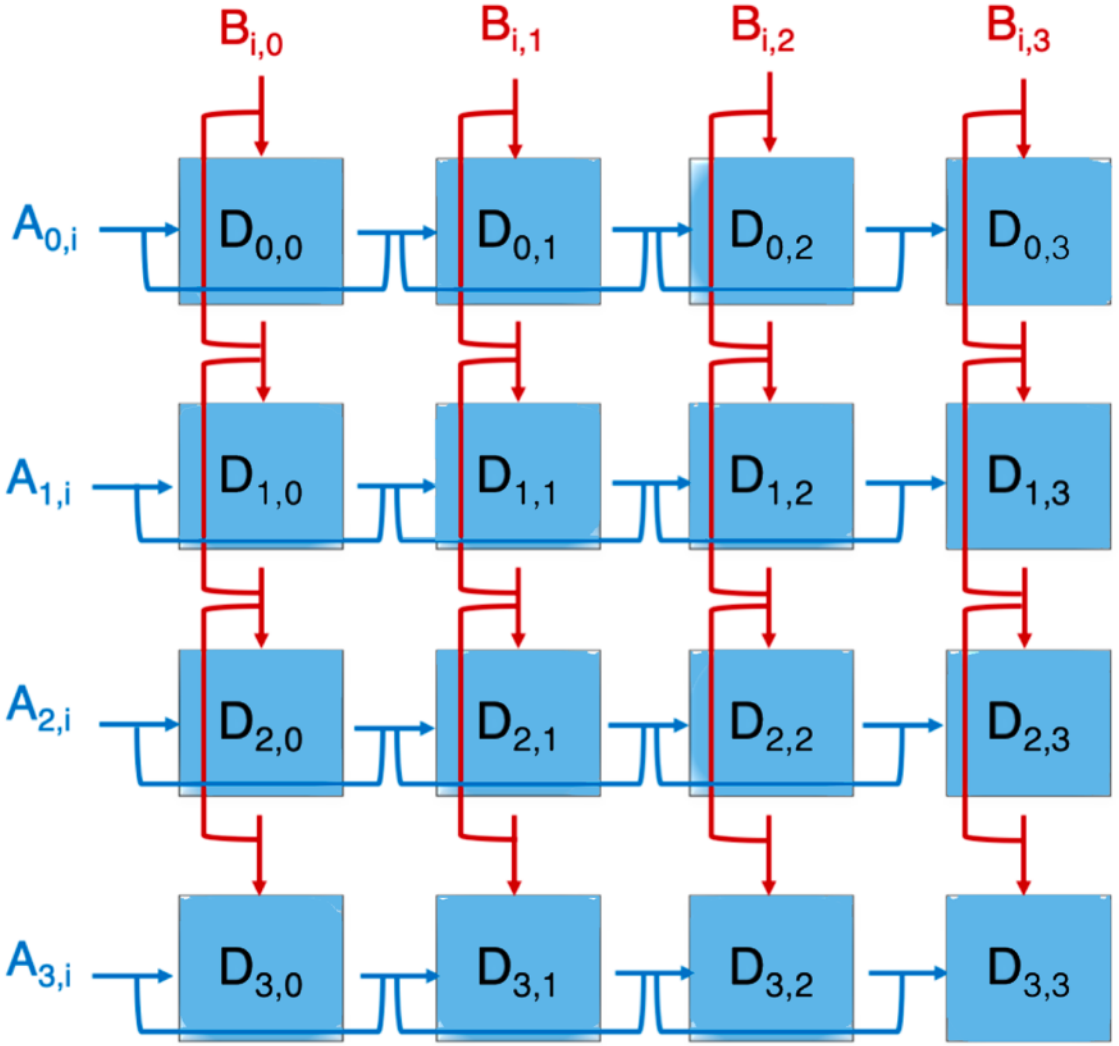


$$D_{0,0} = A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} + A_{0,2} * B_{2,0} + A_{0,3} * B_{3,0} + C_{0,0}$$

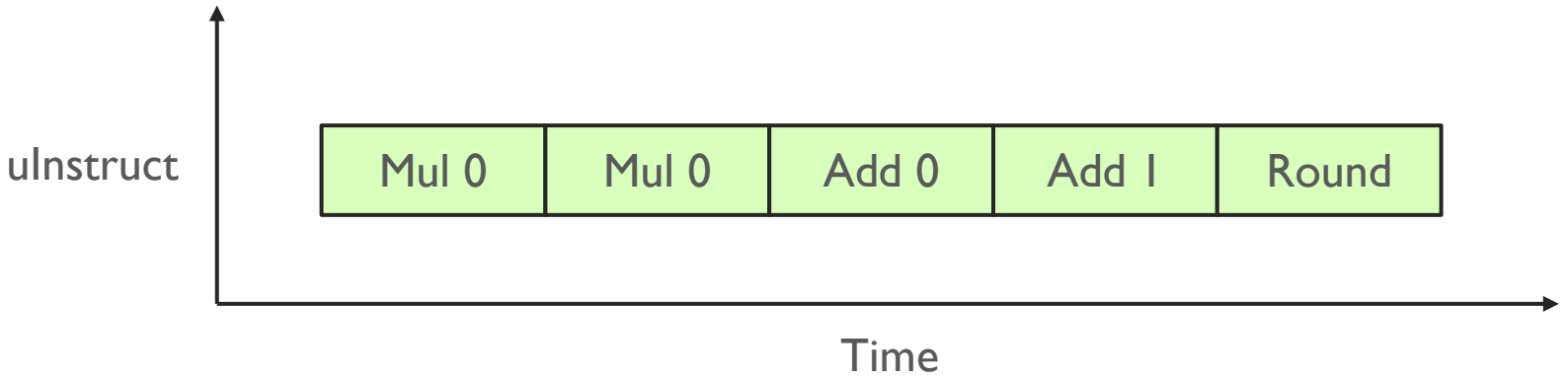
High Level Block Diagram



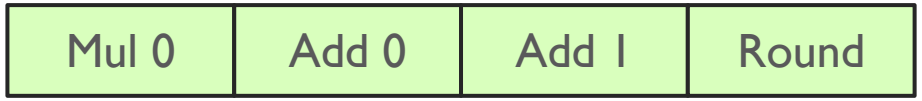
High Level Block Diagram



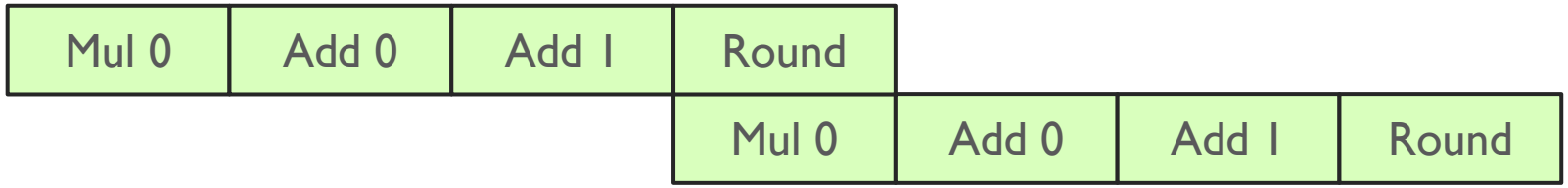
指令流水



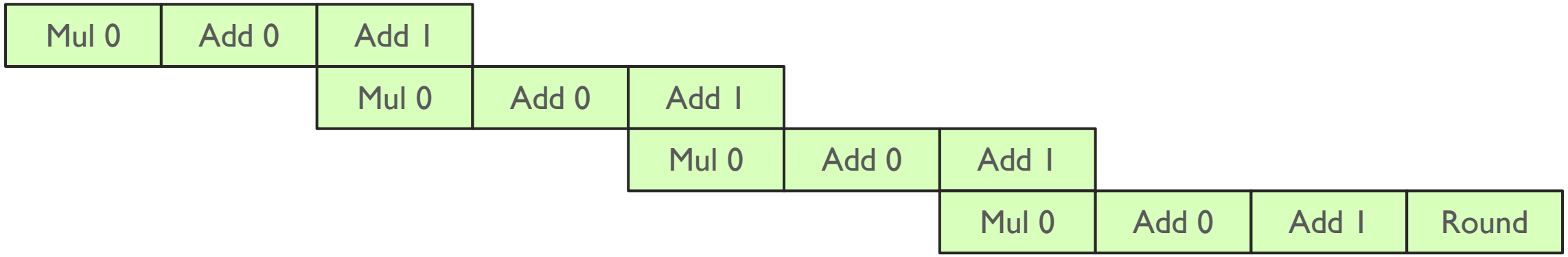
指令流水



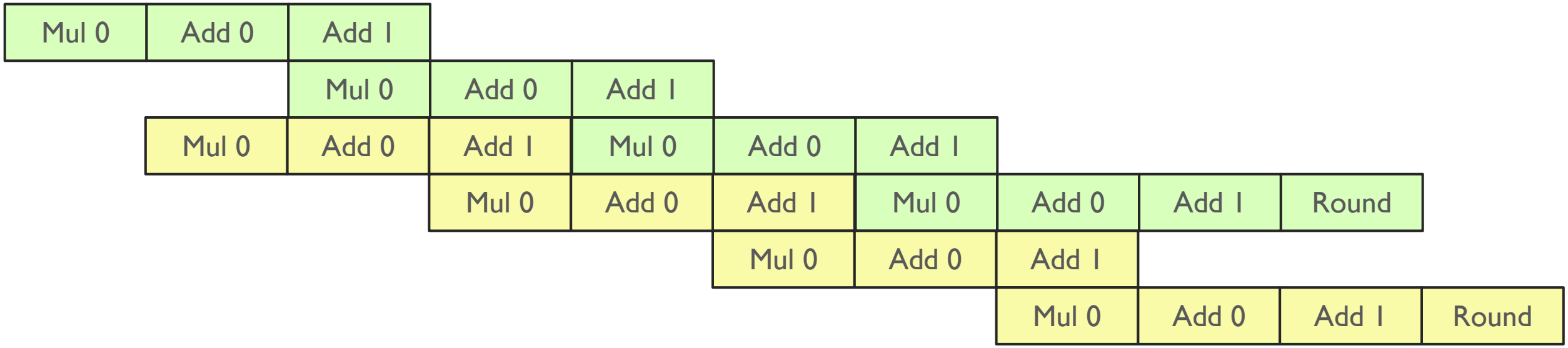
指令流水



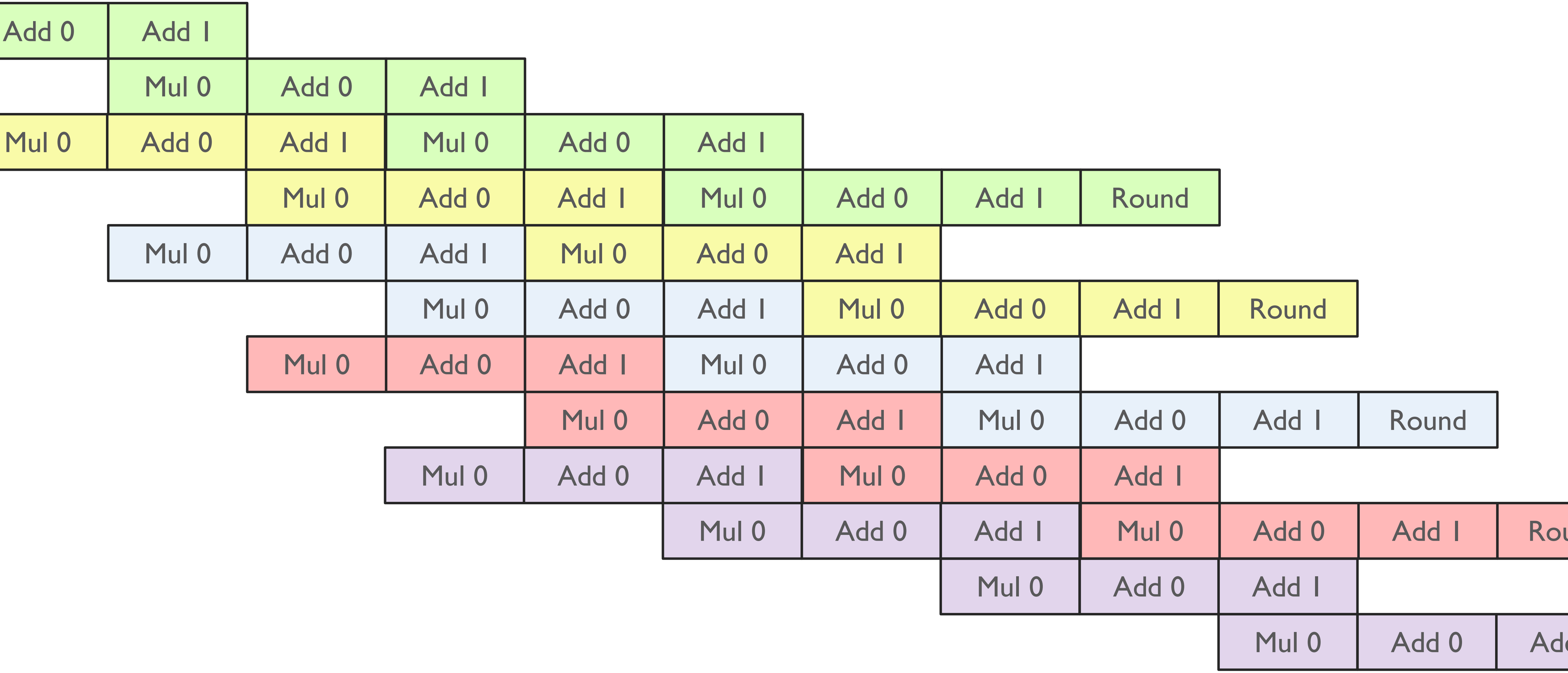
指令流水

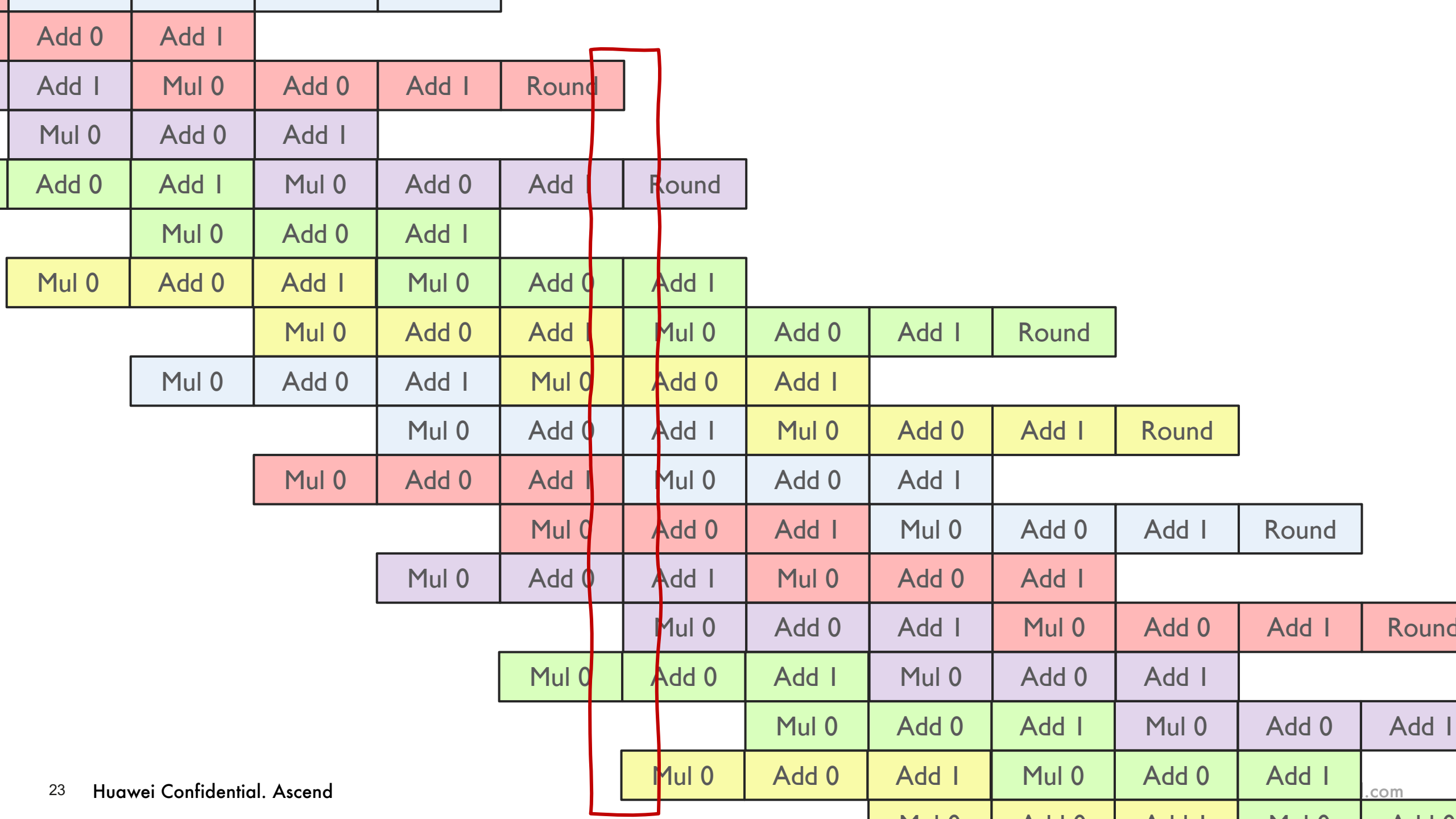

Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Round

指令流水

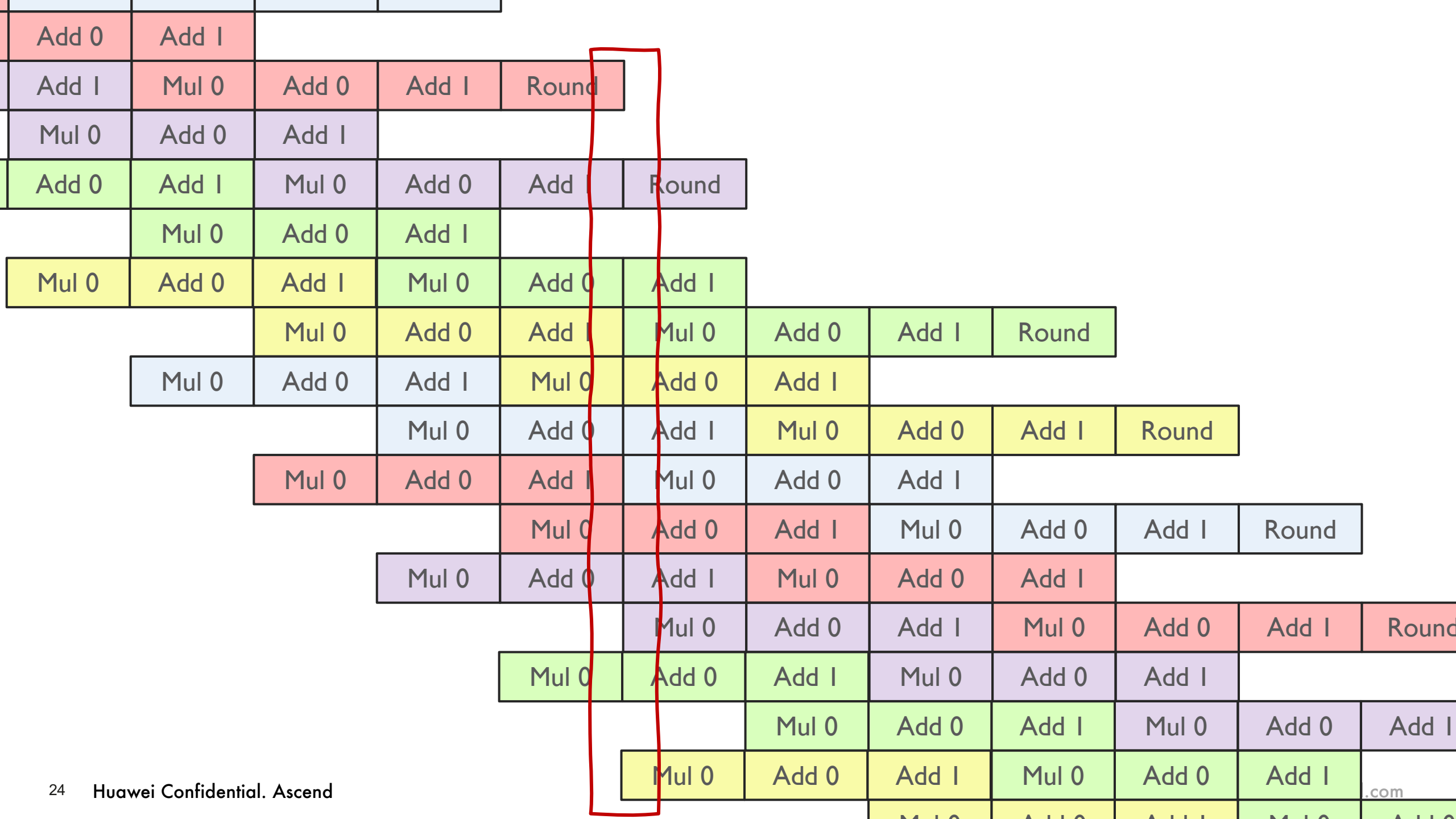

Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Round
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Round

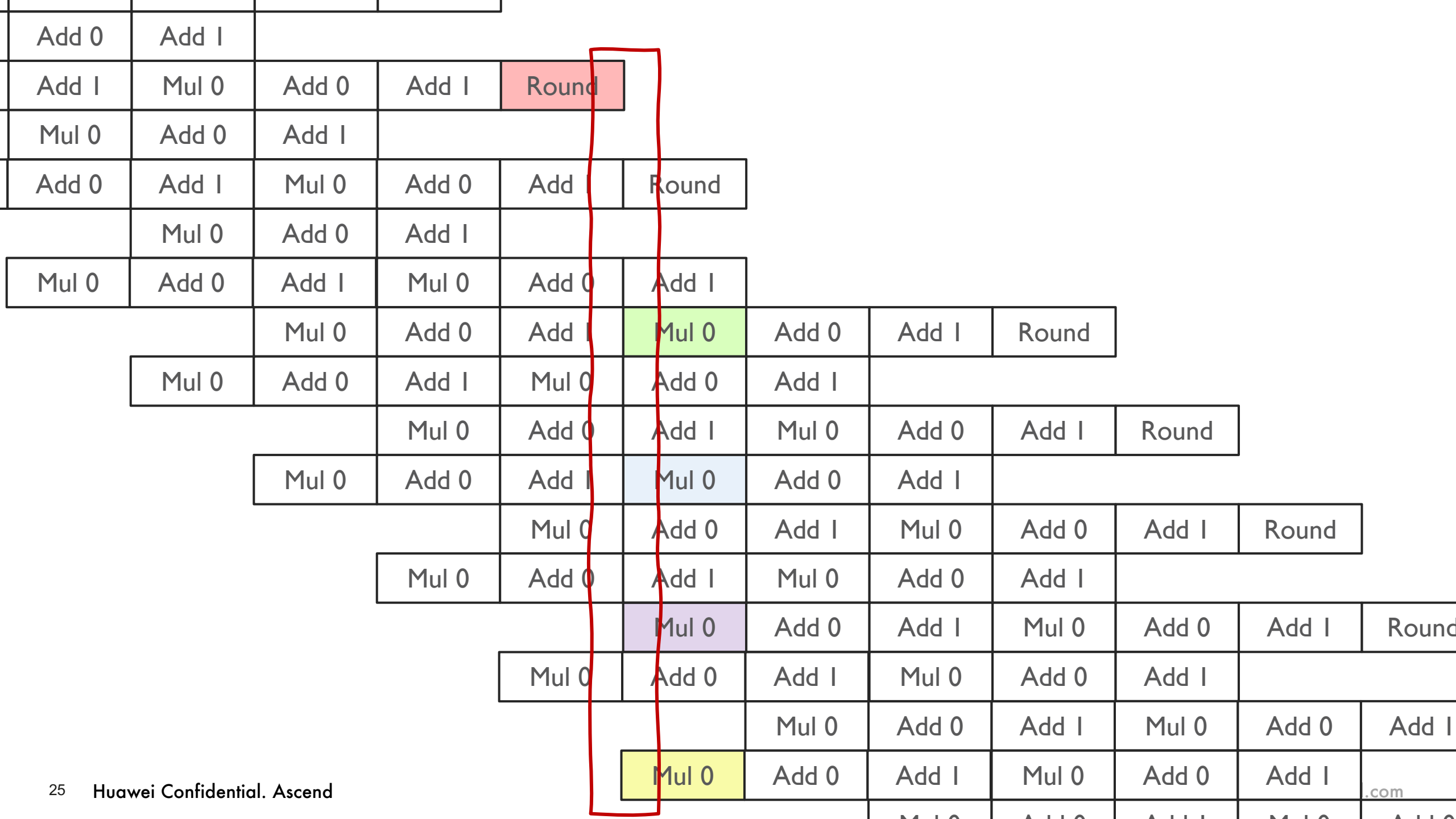
指令流水





Add 0	Add 1										
Add 1	Mul 0	Add 0	Add 1	Round							
Mul 0	Add 0	Add 1									
Add 0	Add 1	Mul 0	Add 0	Add 1	Round						
	Mul 0	Add 0	Add 1								
Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1						
		Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	Round			
	Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1					
			Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	Round		
		Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1				
				Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	Round	
			Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1			
					Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	Round
				Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1		
						Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1
					Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	


Add 0
Add 1
Add 1
Mul 0
Add 0
Add 1
Round
Mul 0
Add 0
Add 1
Add 0
Add 1
Mul 0
Add 0
Add 1
Round
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Round
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Round
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Round
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Rounc
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add I
Mul 0
Add 0
Add 1
Mul 0
Add 0
Add 1

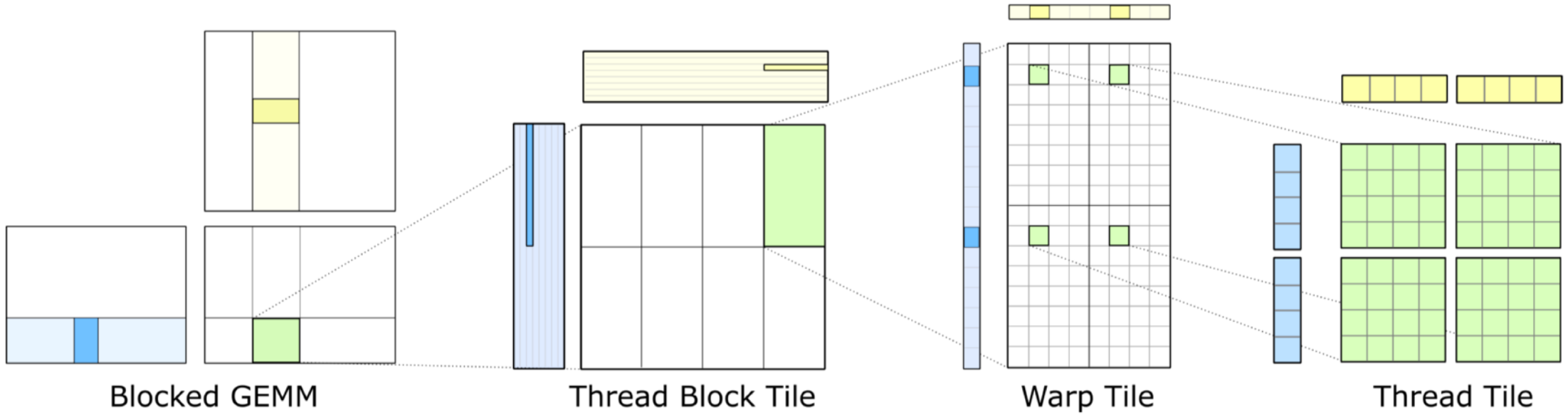


Add 0	Add 1										
Add 1	Mul 0	Add 0	Add 1	Round							
Mul 0	Add 0	Add 1									
Add 0	Add 1	Mul 0	Add 0	Add 1	Round						
	Mul 0	Add 0	Add 1								
Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1						
		Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	Round			
	Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1					
			Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	Round		
		Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1				
				Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	Round	
			Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1			
					Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	Roun
				Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1		
						Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1
					Mul 0	Add 0	Add 1	Mul 0	Add 0	Add 1	

Tensor Core
线程执行

软件设计目标：匹配硬件计算和存储分层结构

- 基于CUDA提供泛型编程 Generic Programing



$$C = A \times B$$

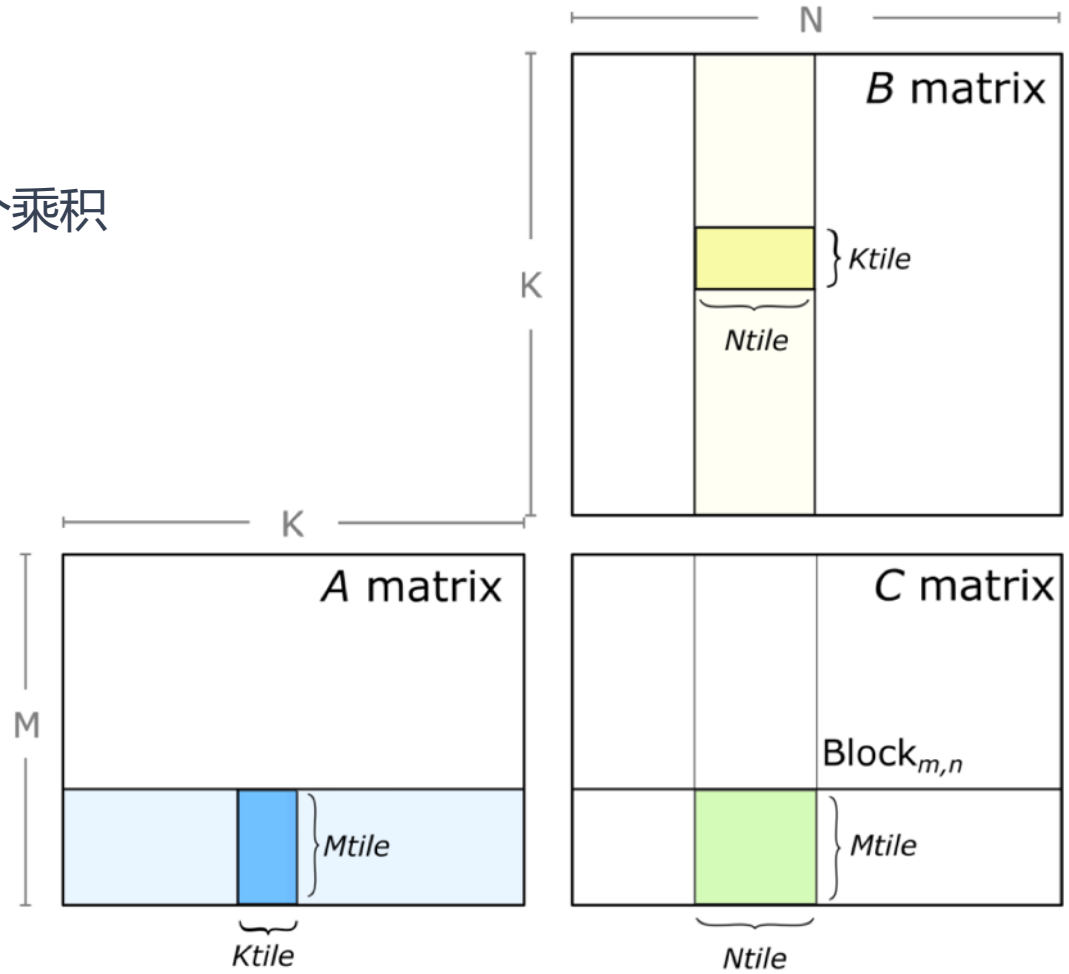
通用的矩阵乘：一次计算一个矩阵块

沿每个维度将循环嵌套 Loop nest 划分为块 blocks

- 划分成为 Mtile-by-Ntile 的独立矩阵乘
- 通过累积 Mtile-by-Ntile-by-Ktile 的矩阵乘积来计算每个乘积

```
for (int mb = 0; mb < M; mb += Mtile)
    for (int nb = 0; nb < N; nb += Ntile)
        for (int kb = 0; kb < K; kb += Ktile)
        {
            // compute Mtile-by-Ntile-by-Ktile matrix product
            for (int k = 0; k < Ktile; ++k)
                for (int i = 0; i < Mtile; ++i)
                    for (int j = 0; j < Ntile; ++j)
                    {
                        int row = mb + i;
                        int col = nb + j;

                        C[row][col] +=
                            A[row][kb + k] * B[kb + k][col];
                    }
        }
```



CUDA中的GEMM：在CUDA线程块中并行

使用 CUDA kernel grid

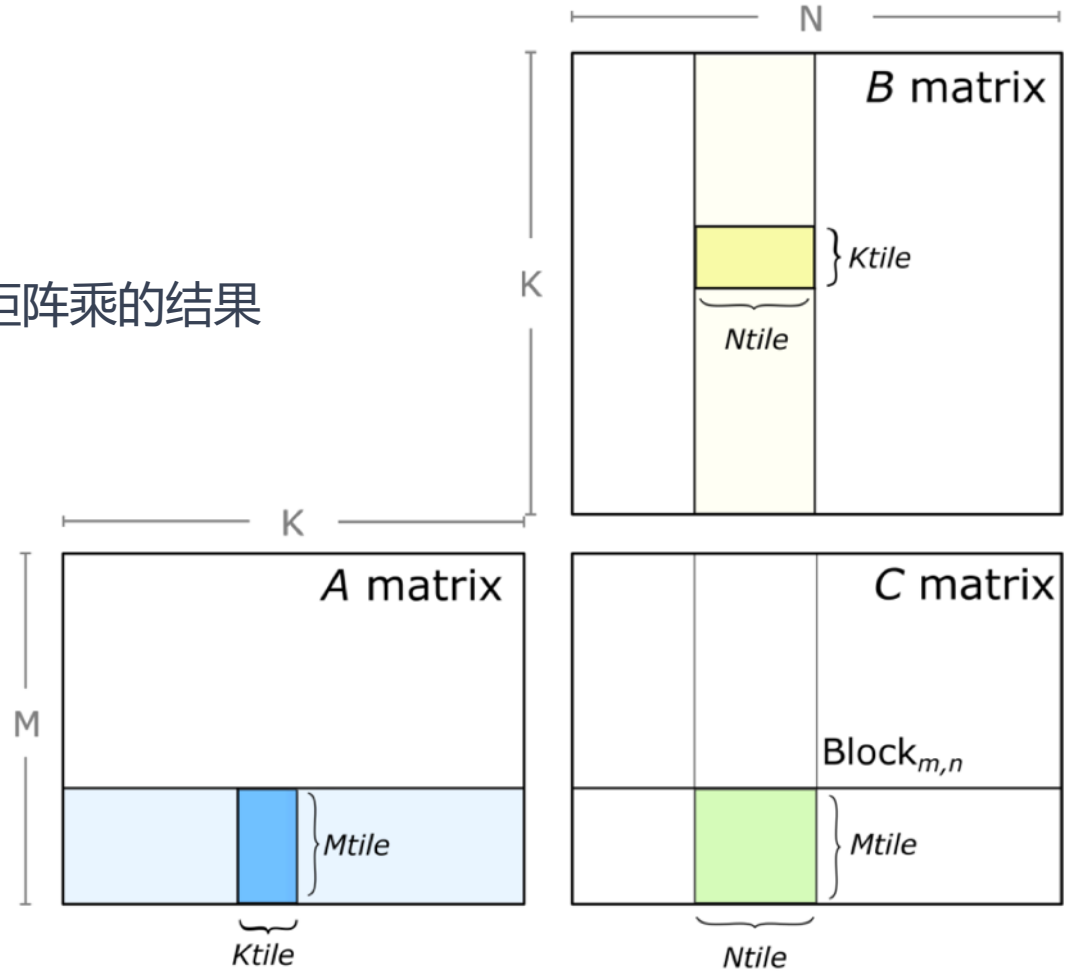
- 将 CUDA 线程块分配给输出矩阵 D 的每个分区

CUDA 线程块并行计算 Mtile-by-Ntile-by-Ktile 矩阵乘

- 在 K 维上进行迭代，执行累积 Mtile-by-Ntile-by-Ktile 矩阵乘的结果

```
for (int mb = 0; mb < M; mb += Mtile)
    for (int nb = 0; nb < N; nb += Ntile)
        for (int kb = 0; kb < K; kb += Ktile)
        {
            .. compute Mtile by Ntile by Ktile GEMM
        }
```

by each CUDA thread block



线程块：CUDA线程块内的并行性

将线程块 Thread block 分解为 warp-level Tiles

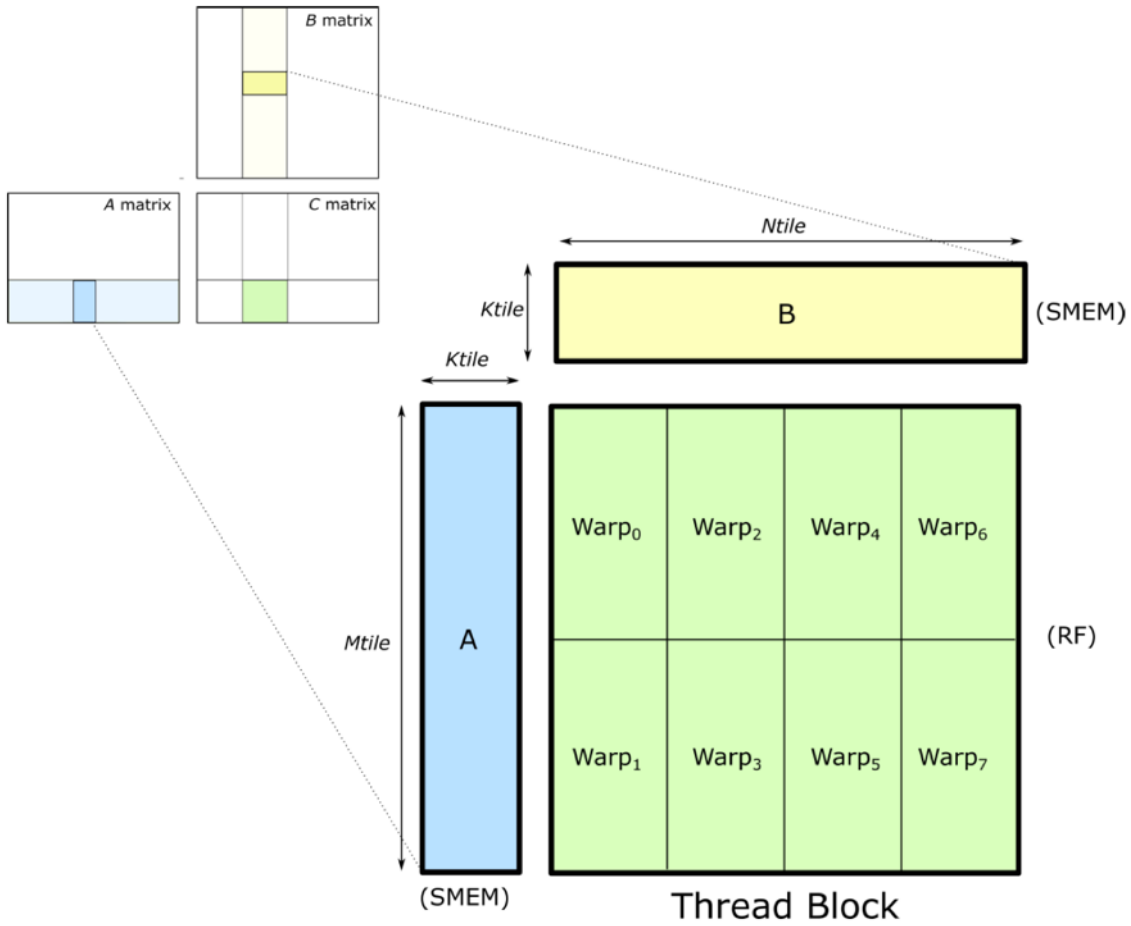
- 把矩阵 A 和 B 加载到可复用的 SMEM 共享内存
- 结果矩阵 C 分布在不同的 Warp 上

每个 warp 计算一个独立的矩阵乘

```
for (int kb = 0; kb < K; kb += Ktile)
{
    .. load A and B tiles to shared memory

    for (int m = 0; m < Mtile; m += warp_m)
        for (int n = 0; n < Ntile; n += warp_n)
            for (int k = 0; k < Ktile; k += warp_k)
                .. compute warp_m by warp_n by warp_k GEMM
}
```

by each CUDA warp



Warp-level 的矩阵乘

Warps-level 来真正执行累积的矩阵乘

- 把 fragment 矩阵 A 和 B 从共享内存加载到寄存器
- SMEM to RF: 数据加载要快于计算，否则阻塞
- 结果矩阵 C 存在线程参与执行的寄存器上

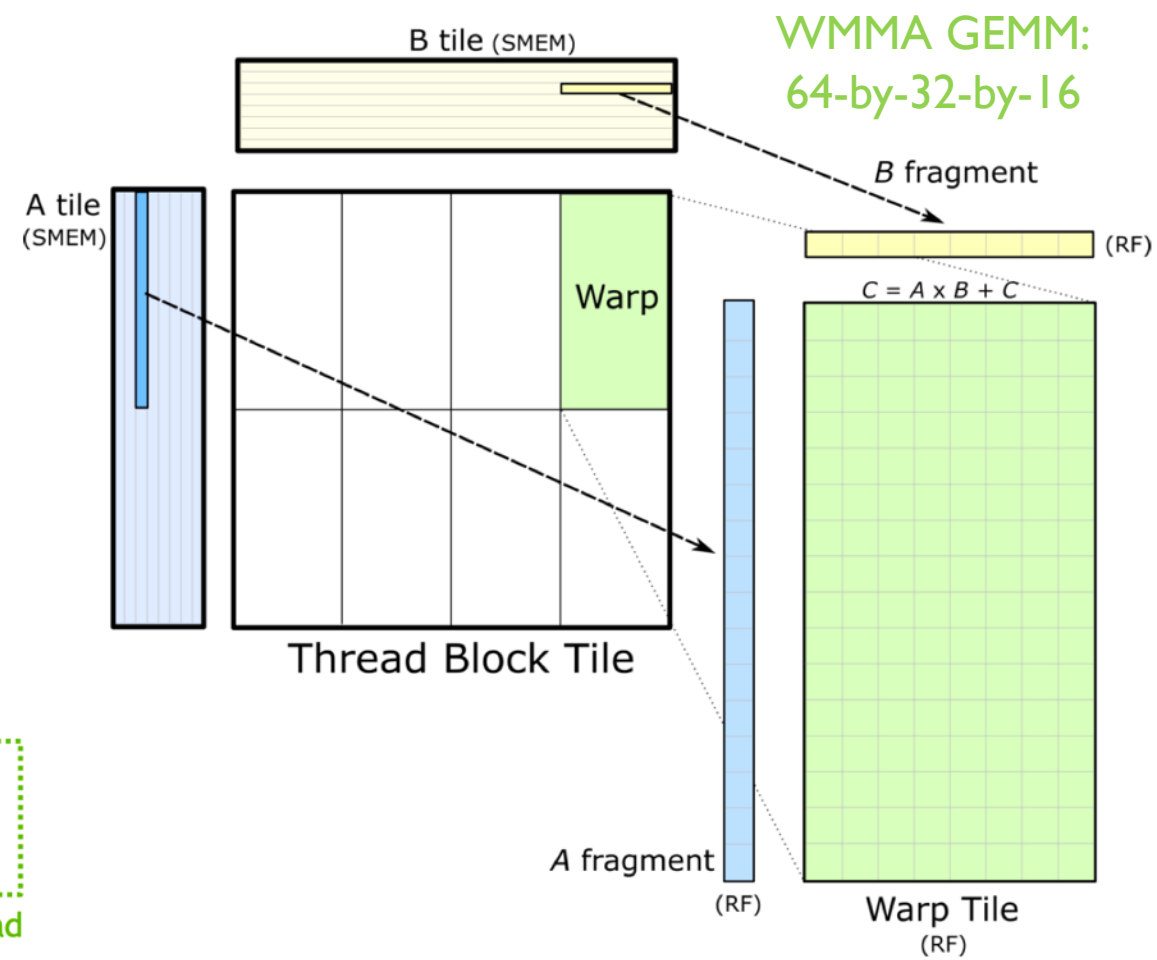
SMEM共享内存 以 K 维排序，线程高效

```
for (int k = 0; k < Ktile; k += warp_k)
{
    .. load A tile from SMEM into registers
    .. load B tile from SMEM into registers

    for (int tm = 0; tm < warp_m; tm += thread_m)
        for (int tn = 0; tn < warp_n; tn += thread_n)

            for (int tk = 0; tk < warp_k; tk += thread_k)
                .. compute thread_m by thread_n by thread_k GEMM
}
```

by each CUDA thread



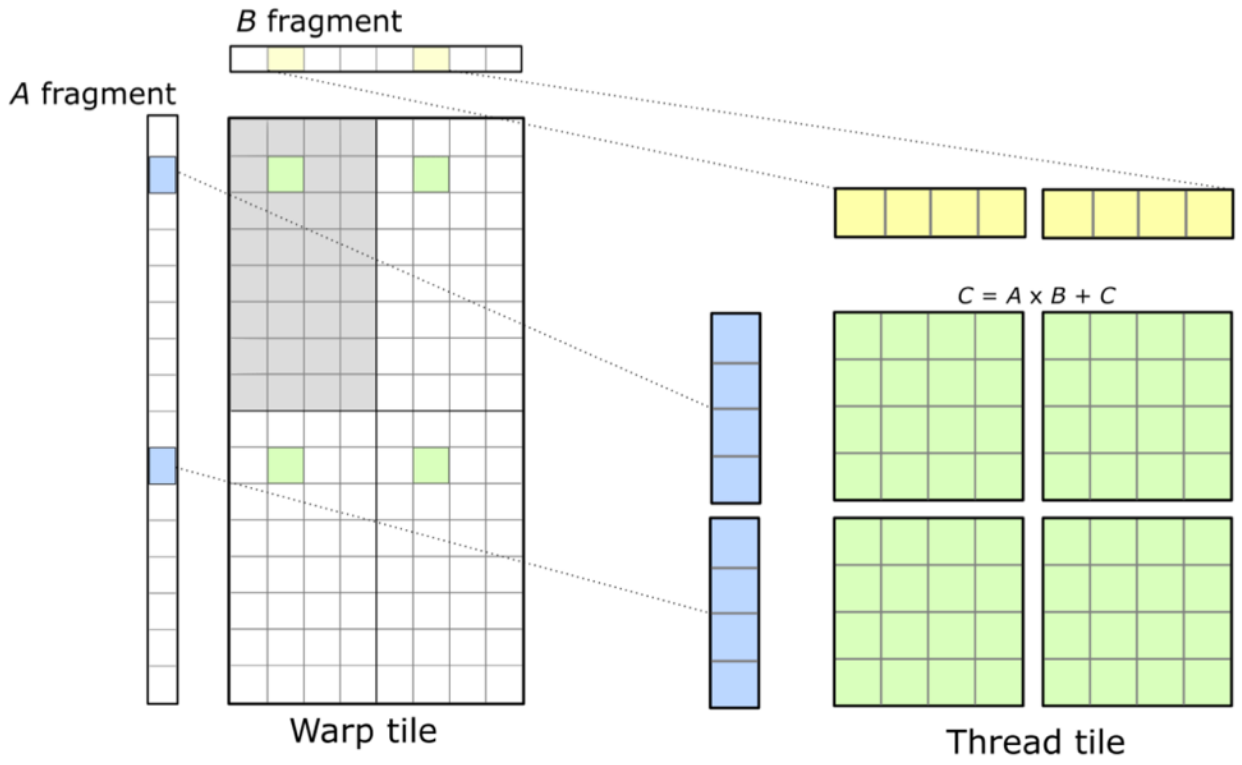
线程-level：Tensor Core上并行执行

Threads 累积的矩阵乘

- A、B、C 矩阵都在寄存器上

```
for (int m = 0; m < thread_m; ++m)
    for (int n = 0; n < thread_n; ++n)
        for (int k = 0; k < thread_k; ++k)
            C[m][n] += A[m][k] * B[n][k];
```

Fused multiply-accumulate instructions



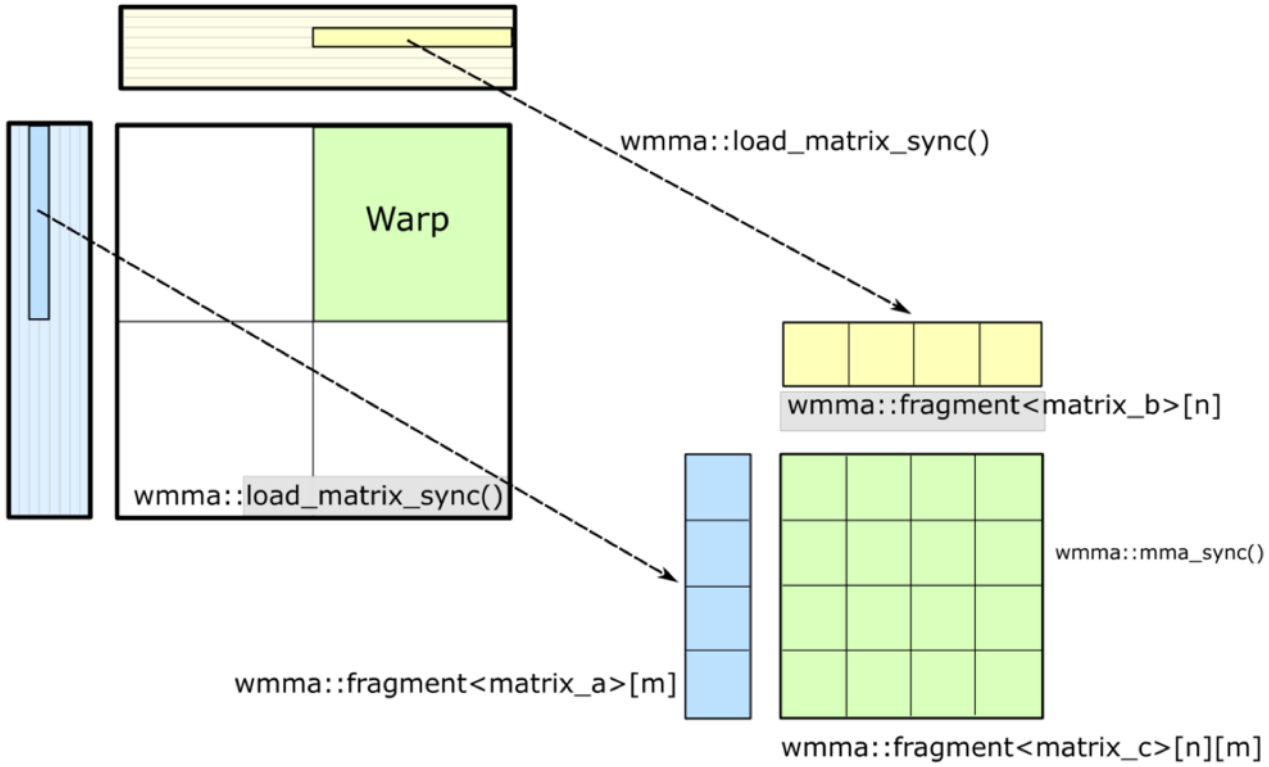
线程-level：Tensor Core上并行执行

Threads 累积的矩阵乘

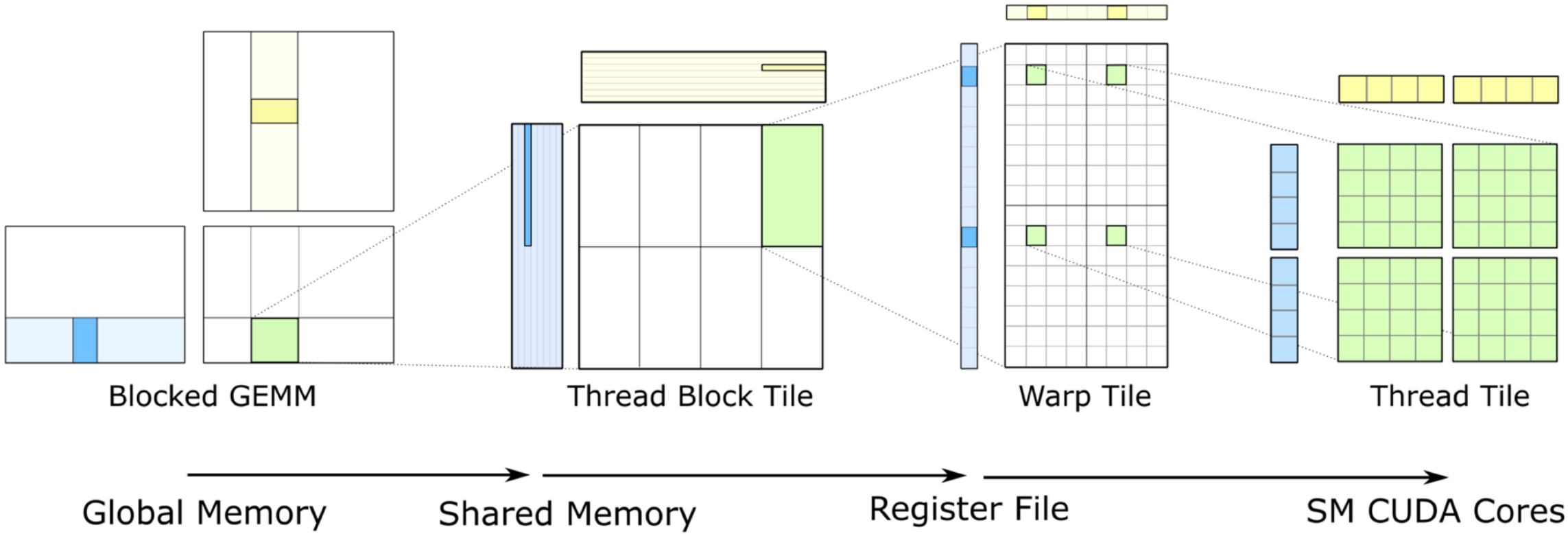
- A、B、C 矩阵都在寄存器上

```
for (int m = 0; m < thread_m; ++m)
    for (int n = 0; n < thread_n; ++n)
        for (int k = 0; k < thread_k; ++k)
            C[m][n] += A[m][k] * B[n][k];
```

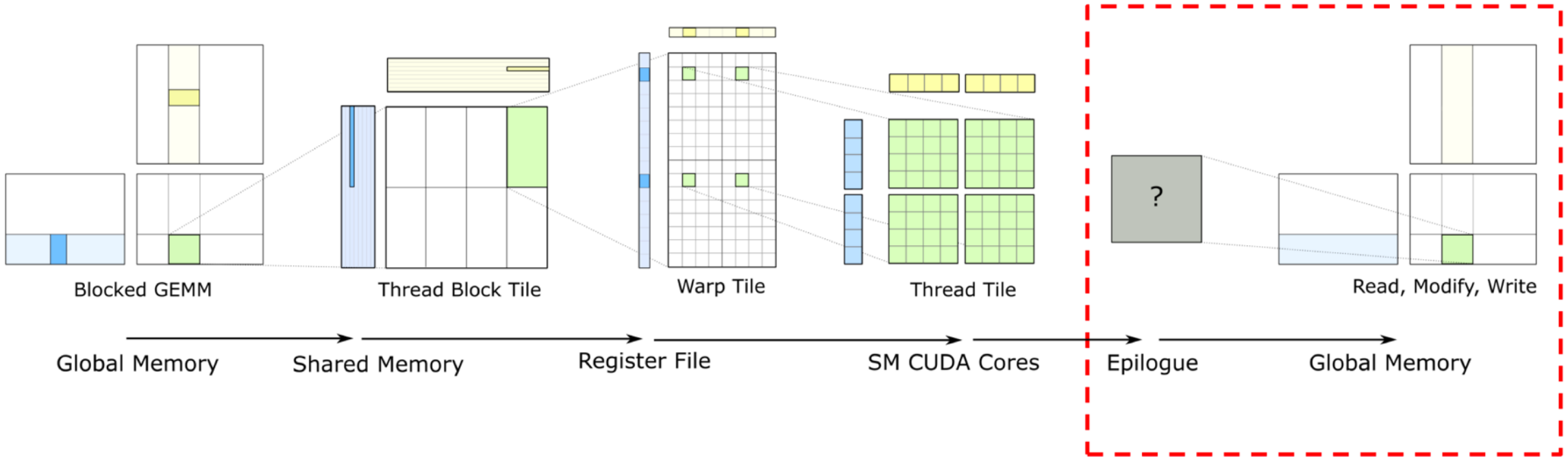
Fused multiply-accumulate instructions



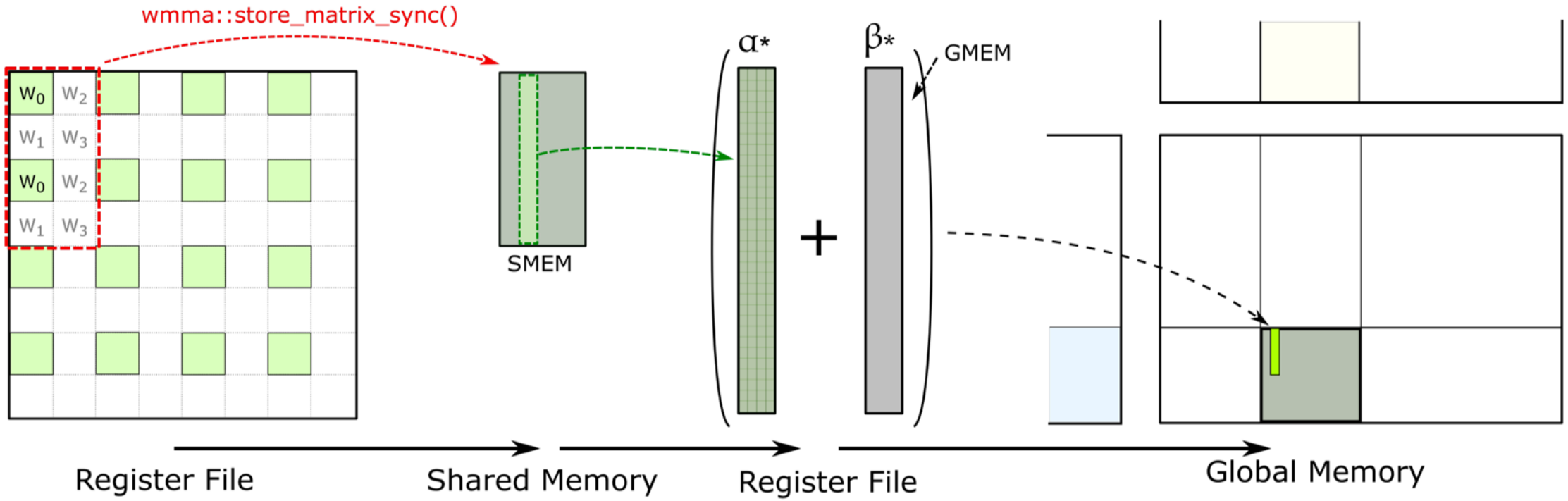
GEMM软硬件分层：每一层都进行数据复用



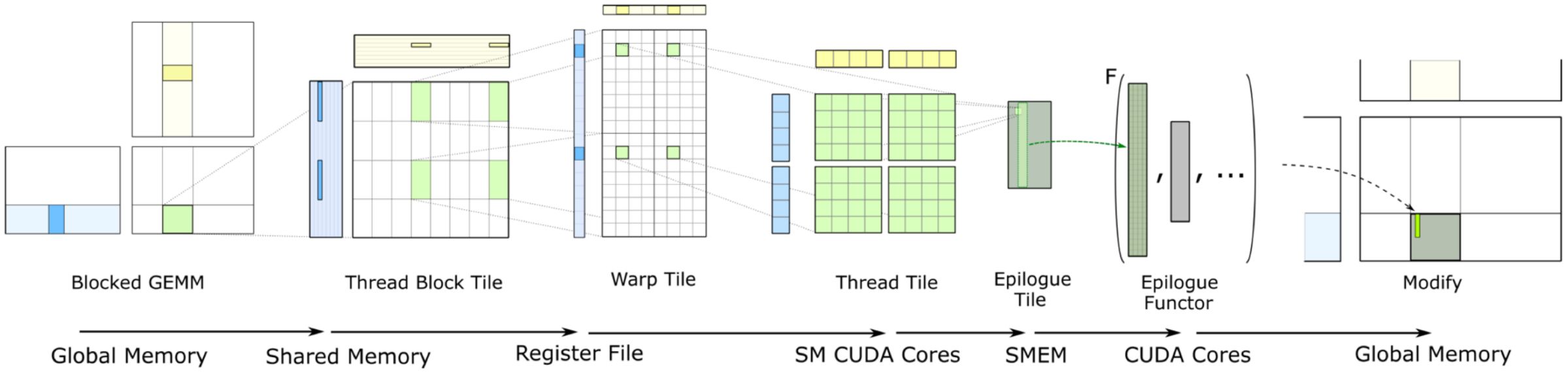
GEMM结束：如何把累积矩阵乘写出最终结果



WMMA 结果回传


wmma::store_matrix_sync()
W0 W2
W1 W3
W0 W2
W1 W3
SMEM
α*
β*
GMEM
Register File
Shared Memory
Register File
Global Memory

完整 GEMM 计算/存储数据流



Reference 引用&参考

1. https://zhuanlan.zhihu.com/p/620257581
2. https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/
3. https://developer.nvidia.com/blog/accelerating-winml-and-nvidia-tensor-cores/
4. https://developer.nvidia.com/blog/optimizing-gpu-performance-tensor-cores/
5. https://www.anandtech.com/Gallery/Album/6494#15
6. https://www.anandtech.com/Gallery/Album/6493#33
7. https://www.anandtech.com/show/12673/titan-v-deep-learning-deep-dive/3
8. https://www.anandtech.com/show/12673/titan-v-deep-learning-deep-dive/4
9. https://www.cnblogs.com/wujianming-110117/p/12993096.html
10. https://aijishu.com/a/1060000000286803#item-2
11. https://learnopencv.com/demystifying-gpu-architectures-for-deep-learning-part-2/