ECE408 / CS483 / CSE408
Summer 2025

Applied Parallel Programming

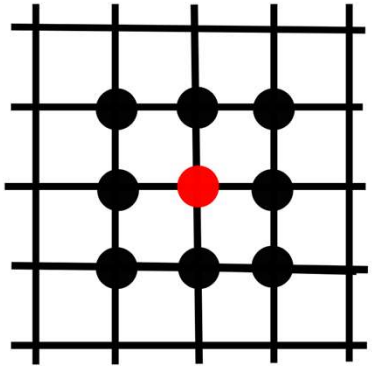
Lecture 10:
Tiled Convolution Analysis

What Will You Learn Today?

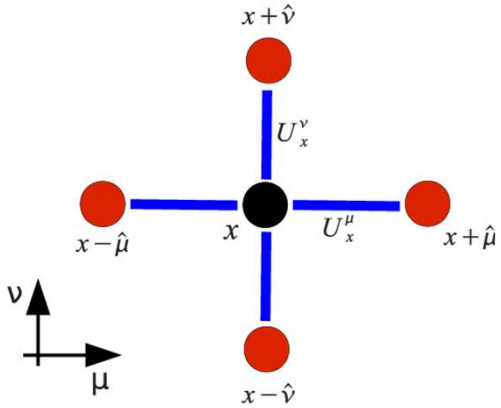
- more detailed analysis of tiled convolution/stencil algorithms

Stencil Algorithms

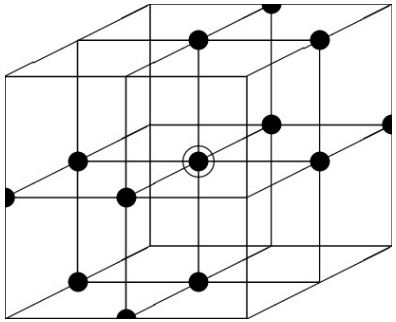
- Numerical data processing algorithms which update array elements according to some fixed pattern, called a *stencil*
  - Convolution is just one such example



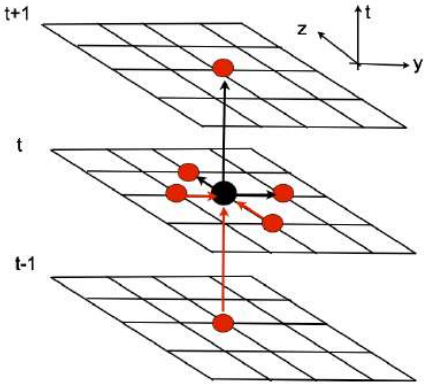
2D convolutional kernel (9-point 2D stencil)



Nearest neighbor lattice Dirac operator (5-point 2D stencil)

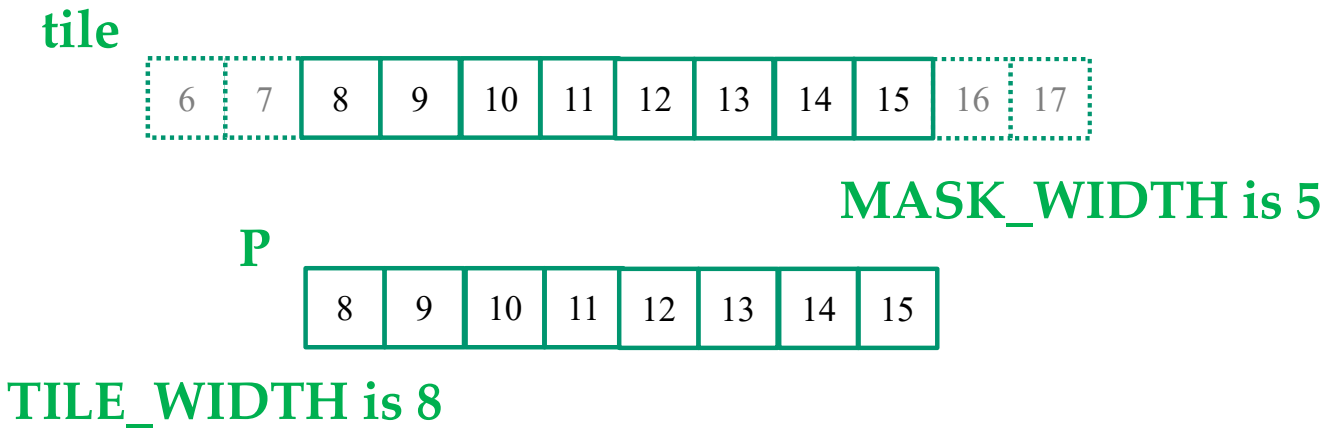


Finite difference stencil for 3D explicit time-marching (13-point 3D stencil)



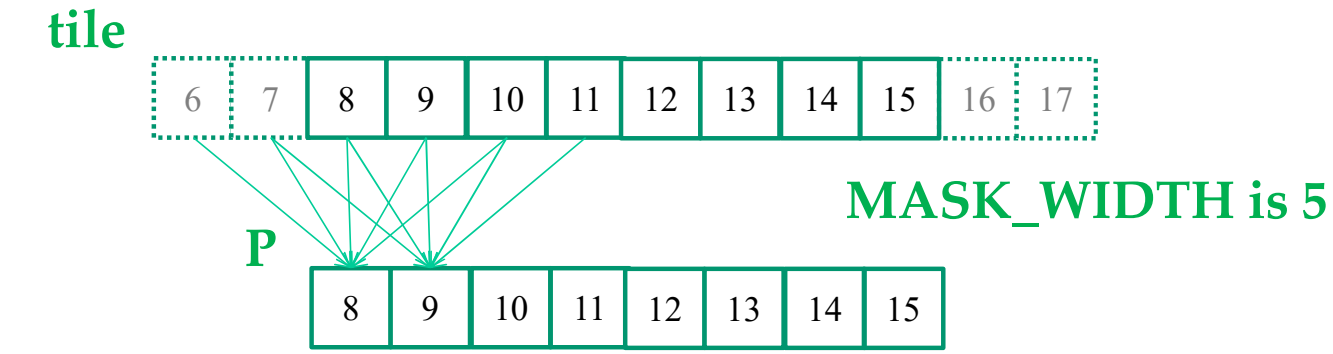
The Wilson-Dslash operator (4D stencil)

A Small 1D Example



- output and input tiles for block 1
- For MASK_WIDTH of 5, each block loads 8 + (5 – 1) = 12 elements (**12 memory loads**)

Each Output Uses MASK_WIDTH Inputs



TILE_WIDTH is 8

- P[8] uses N[6], N[7], N[8], N[9], N[10]
- P[9] uses N[7], N[8], N[9], N[10], N[11]
- …
- P[15] uses N[13], N[14], N[15], N[16], N[17]

Total of **8×5 values** from **`tile`** **used for** the **output**.

A simple way to calculate tiling benefit

- **(8+5-1) = 12** elements loaded
- **8×5 = 40** global memory accesses replaced by shared memory accesses
- Bandwidth reduction of **40/12 = 3.3**

(ignoring boundary effects, so reached asymptotically for large **N** inputs)

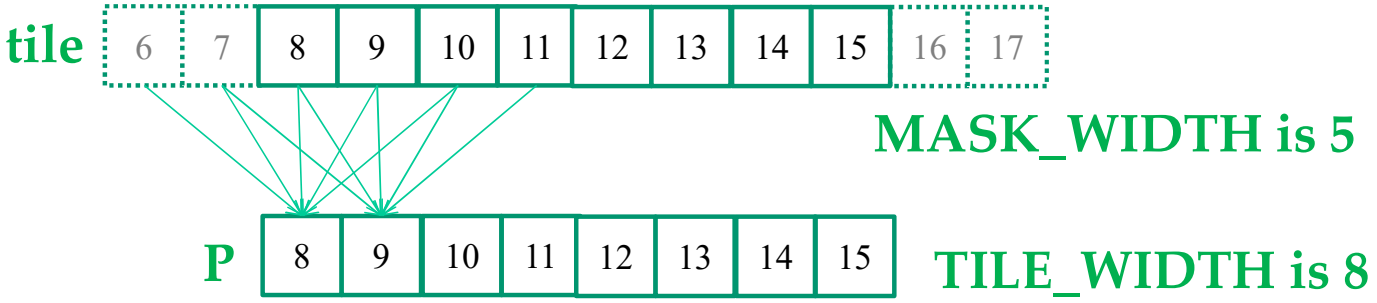
In General, for 1D

- Load **TILE_WIDTH + MASK_WIDTH - 1** elements.
- Replace **TILE_WIDTH × MASK_WIDTH** global memory accesses with shared memory.

- Bandwidth reduction of

(TILE_SIZE×MASK_WIDTH)/(TILE_SIZE+MASK_WIDTH-1)

Another Way to Look at Reuse



- tile[6] is used by P[8] (1×)
- tile[7] is used by P[8], P[9] (2×)
- tile[8] is used by P[8], P[9], P[10] (3×)
- tile[9] is used by P[8], P[9], P[10], P[11] (4×)
- tile[10] is used by P[8], P[9], P[10], P[11], P[12] (5×)
- … (5×)
- tile[14] is uses by P[12], P[13], P[14], P[15] (4×)
- tile[15] is used by P[13], P[14], P[15] (3×)
- tile[16] is used by P[14], P[15] (2×)
- tile[17] is used by P[15] (1×)

Another Way to Look at Reuse

- Each access `tile` replaces an access to **N**.
- **Global** memory **accesses**
  - **to (8 + 5 - 1) = 12 elements** of N
  - **replaced by shared** memory **accesses** is

1 + 2 + 3 + 4 + 5 × (8 – 5 + 1) + 4 + 3 + 2 + 1

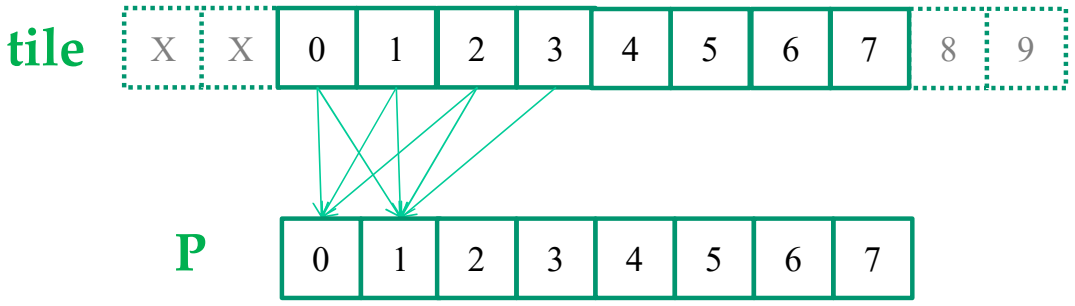
= 10 + 20 + 10

= 40

There are **12** elements of **N**, so the average reduction is

40/12 = 3.3

What about Boundary Tiles?



- P[0] uses N[0], N[1], N[2]
- P[1] uses N[0], N[1], N[2], N[3]
- P[2] uses N[0], N[1], N[2], N[3], N[4]

Fewer than 8 × 5 elements of **N** are **used** for the output tile!

Ghost Elements Change Ratios

- For a boundary tile, we load

  TILE_WIDTH + (MASK_WIDTH-1)/2 elements

  – **10** in our example of **TILE_WIDTH** of **8**

  – and **MASK_WIDTH** of **5**.

- Computing boundary elements

  – do not access shared memory for ghost cells, so

  – total accesses is **6×5 + 4 + 3 = 37** accesses

  – when computing the **P** elements.

The reduction is **37/10 = 3.7** .

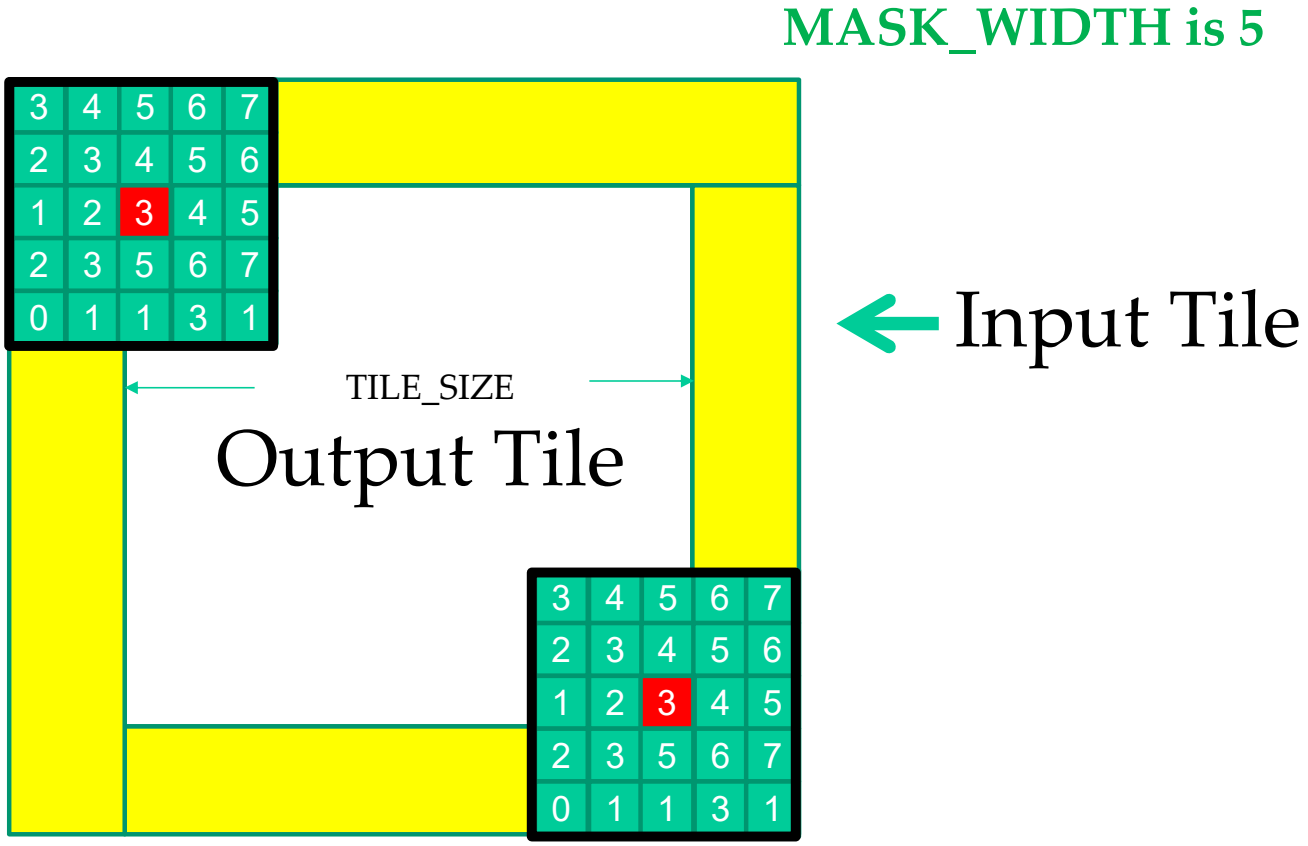
Bandwidth Reduction for 1D

- The reduction is

(TILE_SIZE*MASK_WIDTH)/(TILE_SIZE+MASK_WIDTH-1)

TILE_WIDTH	16	32	64	128	256
Reduction Mask_Width = 5	4.0	4.4	4.7	4.9	4.9
Reduction Mask_Width = 9	6.0	7.2	8.0	8.5	8.7

Review: Parallelization of Tile Load



8×8 Output Tile, MASK_WIDTH of 5

- Loading input tile requires **$(8+5-1)^2 = 144$ reads.**
- Calculation of each output requires **$5^2 = 25$** input elements.
- **8×8×25 = 1,600 global memory accesses** for computing output tile converted to shared memory accesses.
- Bandwidth reduction of **1,600/144 = 11.1×**

In General

- **(TILE_WIDTH+MASK_WIDTH-1)^2 elements** need to be **loaded** from **N** into shared memory
- The calculation of each **P** element needs to access **MASK_WIDTH^2** elements of **N**
- **TILE_WIDTH^2 × MASK_WIDTH^2 global memory accesses** converted into shared memory accesses

- Bandwidth reduction of

TILE_WIDTH^2×MASK_WIDTH^2/(TILE_WIDTH+MASK_WIDTH-1)^2

Bandwidth Reduction for 2D

- The reduction is

$$\text{TILE\_WIDTH}^2 \times \text{MASK\_WIDTH}^2 / (\text{TILE\_WIDTH} + \text{MASK\_WIDTH} - 1)^2$$

TILE_WIDTH	8	16	32	64
Reduction Mask_Width = 5	11.1	16	19.7	22.1
Reduction Mask_Width = 9	20.3	36	51.8	64

2B/FLOP for Untiled Convolution

How much global memory per FLOP in untiled convolution?

In untiled convolution,

- each value from **N** (**4B** from global memory)
- is multiplied by a value from **M** (**4B** from constant cache, **1 FLOP**),
- then added to a running sum (**1 FLOP**).

That gives 2B / FLOP.

Full Use of Compute Requires 13.3× Reuse

Recall our reuse discussion from matrix multiply:

- **1,000 GFLOP/s** for GPU from ~2010, and
- **150 GB/s** memory bandwidth.

Dividing memory bandwidth by **2B/FLOP**,

$$\frac{150\ \text{GB/s}}{2\ \text{B/FLOP}} = \mathbf{75\ GFLOP/s}$$ **= 7.50% of peak**.

Need at least **100/7.50 = 13.3× reuse**
to make full use of compute resources.

In 2020, Need 52.1× Reuse

That was 2010.

In 2020, the **GRID K520** (remember MP0?) offers

- nearly **5,000 GFLOP/s**, but only
- **192 GB/s** memory bandwidth.

Dividing memory bandwidth by **2B/FLOP**,

$$\frac{192\ \text{GB/s}}{2\ \text{B/FLOP}} = \mathbf{96\ GFLOP/s}$$ **= 1.92% of peak**.

Need at least **100/1.92 = 52.1× reuse**
to make full use of compute resources.

Need ~26× Reuse on H100 GPUs

- **In 2023**, the **H100 PCIe** version offers
  - **26 TFLOP/s**, and
  - **2 TB/s** memory bandwidth

- Dividing memory bandwidth by **2B/FLOP**,

$$\frac{2\ \text{TB/s}}{2\ \text{B/FLOP}} = \mathbf{1\ TFLOP/s = 3.85\%\ of\ peak}$$

- **Need** at least **100/3.85 = 25.97× reuse** to make full use of compute resources.

Need Really Big Mask to Balance Resources

- Let's make another table: **% of peak compute**
  - for **1D** tiled convolution,
  - with **TILE_WIDTH 1024**

MASK_WIDTH	~2010 GPU with 1,000 GFLOP/s 150 GB/s	~2020 GPU with 5,000 GFLOP/s 192 GB/s
5	37%	9.6%
9	67%	17%
15	100%	28%
55	100%	100%

Need Really Big Mask to Balance Resources

- And one more: **% of peak compute**
  - for **2D** tiled convolution,
  - with **TILE_WIDTH 32×32**

MASK_WIDTH	~2010 GPU with 1,000 GFLOP/s 150 GB/s	~2020 GPU with 5,000 GFLOP/s 192 GB/s
3	60%	15%
5	100%	37%
7	100%	67%
9	100%	almost 100%

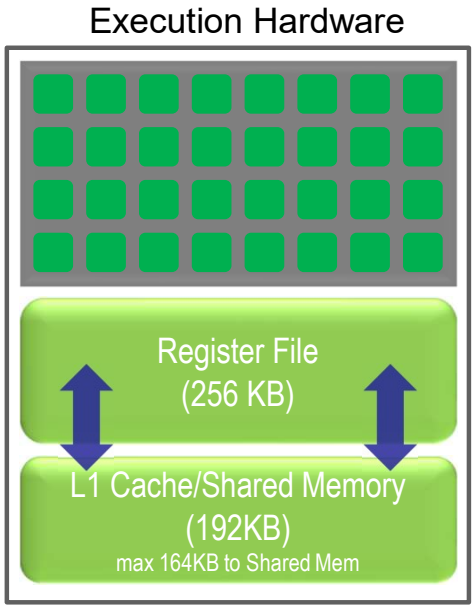
Food for Thought

- Ratios are different for tiles on boundaries.
- More importantly,
  - Each thread loads 4B to shared memory.
  - 2,048 threads load only 8kB.
  - Shared memory is usually 64kB or larger.
  - What can one do with the rest?

Improved approach left as homework.

(For example, can raise MW=7 from 67% to 81%.)

Ampere SM Memory Architecture

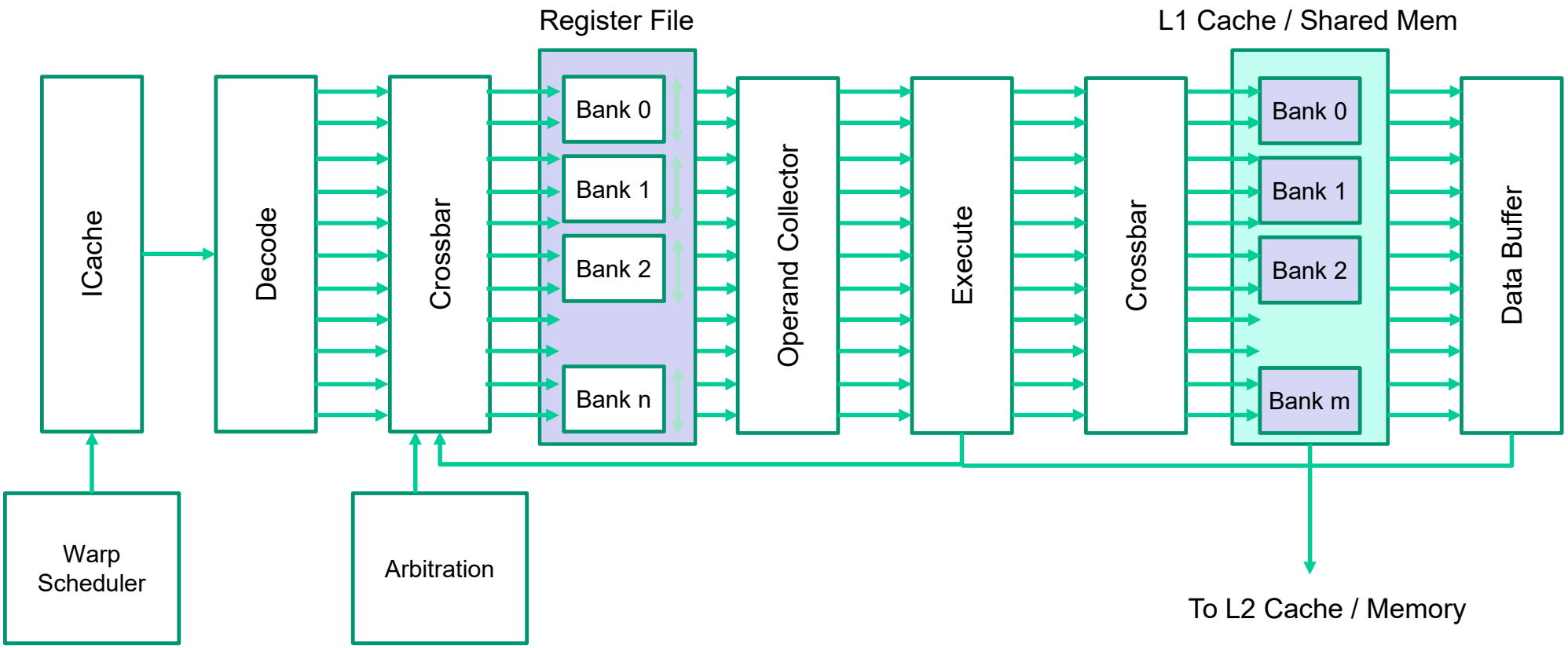


Ampere SM Memories
(GPU from 2020)

- **Memory access time**
  - **registers (~1 cycle)**
  - **shared memory (~5 cycles)**
  - **cache/constant memory (~5 cycles)**
  - **global memory (~500 cycles)**

- **Number of thread blocks mapped to an SM is limited by**
  - **Total number of threads supported by an SM**
  - **Amount of shared memory available on SM**

Overall Data Parallel Pipeline



Memory Hierarchy Considerations

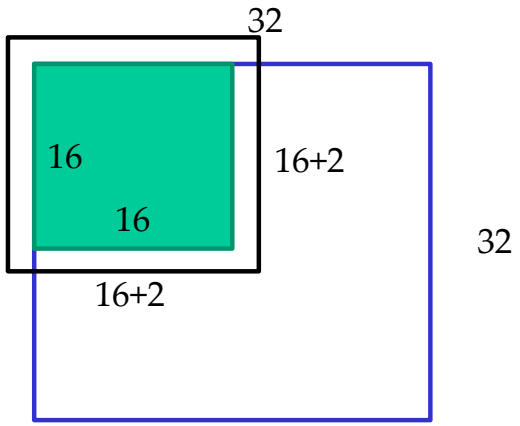
- Register file is highly banked
  - But we can have bank conflicts that cause pipeline stalls
- Shared memory is highly banked
  - But we can have bank conflicts that cause pipeline stalls
- Global memory has multiple channels, banks, pages
  - Relies on bursting
  - Coalescing is important. Need programmer involvement.
- L1 Cache is non-coherent

QUESTIONS?

READ CHAPTER 7!

Problem Solving

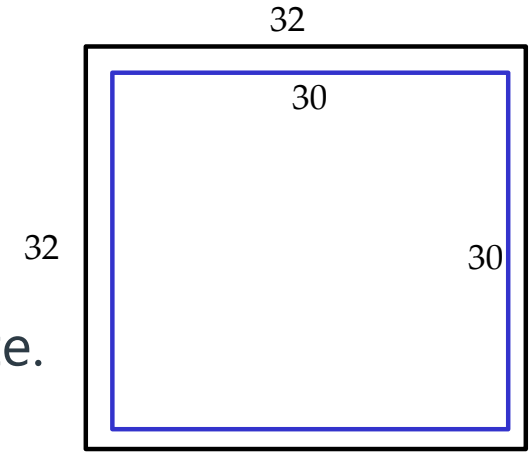
- Q: Consider 2D tiled convolution using output parallelism and multiple loads to bring the full input tile into shared memory. Mask width is 5x5 and output tile width is 16x16, applied to an input image of 32x32. What is the reduction in global memory accesses for thread block (0,0)?
- A:
  - Count # of global memory accesses:
    row 0: 9 + 12 + 15×14
    row 1: 12 + 16 + 20×14
    other 14 rows: 15 + 20 + 25×14
  - Count # of input elements to be used in the calculation: 18×18



Problem Solving

Q: For a tiled 2D convolution kernel using input parallelism, 30x30 output tiles, and 3x3 mask (thus 32x32 input tiles), how many warps in each thread block have control divergence for a 30x30 input?

A: thread block size is 32x32



- All threads participate in data load.
- Threads with tx < 30 && ty < 30 compute.
- What does each warp do?
  - Warp 0: loads 0s, 30 threads compute.
  - Warps 1-29: loads 0s/data, 30 threads compute.
  - Warp 30: loads 0s/data, does not compute.
  - Warp 31: load 0s, does not compute.
- Thus, only 1 warp has no control divergence (2 warps, #30 and #31, for an interior tile in a large input).

Problem Solving

Q: The A40 GPU has a peak FP32 performance of 37.4 TFLOPS and 48 GB of GDDR6 memory with a memory bandwidth of 696 GB/s. These GPUs do not support FP64. What should be the byte-to-FLOP ratio for this GPU to fully utilize its compute capability? How much of data reuse is needed to make full use of the compute resources?

A: 696 GB/s / 37.4 TFLOPS = 0.019 B/FLOP

Another way to look at it: for each byte loaded from global memory, we need to use it in 53.7 floating point operations.