Midterm Review

UC RIVERSIDE
UNIVERSITY OF CALIFORNIA

Logistics

- Lab 2 now due Monday May 18th
- Midterm next class
  - computer architecture background, gpu architecture, CUDA Parallelism, Memory coalescing, warp divergence, thread synchronization, Reduction, Scan, and Matrix Multiplication parallel algorithms
- UCR Cares Act
  - Hopefully, you have received an email from the financial aid office about receiving your CARES Act fund
  - Sign up for direct deposit through your student account in rweb

Quiz 2 – Question 1

- Allocate
  - cudaMalloc((void**) &d_img, sizeof(float)*width*height);
  - Do not allocate height and width as they are not pointers
- Copy to device
  - cudaMemcpy(d_img, h_img, sizeof(float)*width*height, cudamemcpyHostToDevice);
  - Destination, source, size, direction
- Launch
  - BlockDim = (32,32,1) – given in question
  - GridDim = (ceil(width/32),ceil(height/32),1) – gridDim also needs to be 2D
  - ProcessImage<<<gridDim,BlockDim>>>(d_img,height,width)
- Copy to host
  - cudaMemcpy(h_img, d_img, sizeof(float)*width*height, cudamemcpyDeviceToHost);

Quiz 2 – Question 2

- This does not exhibit coalesced memory requests
- Coalesced requests follow indexing pattern of
- [a + tid.x] where a is some independent expression

```
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;
// get 1D coordinate for the grayscale image
int grayOffset = y*width + x;
// one can think of the RGB image having
// CHANNEL times columns than the gray scale image
int rgbOffset = grayOffset*CHANNELS;
```

- rgbOffset does not follow this pattern
- [(a+x)*CHANNELS]

Quiz 2 – Question 2 uncoalesced

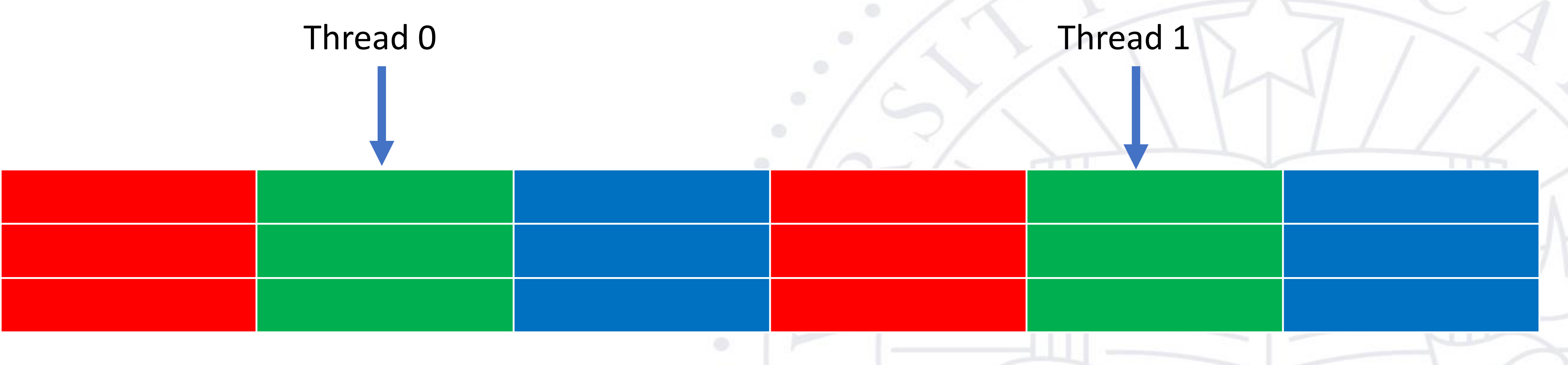
unsigned char r = rgbImage[rgbOffset]; *// red value for pixel*

Thread 0

Thread 1

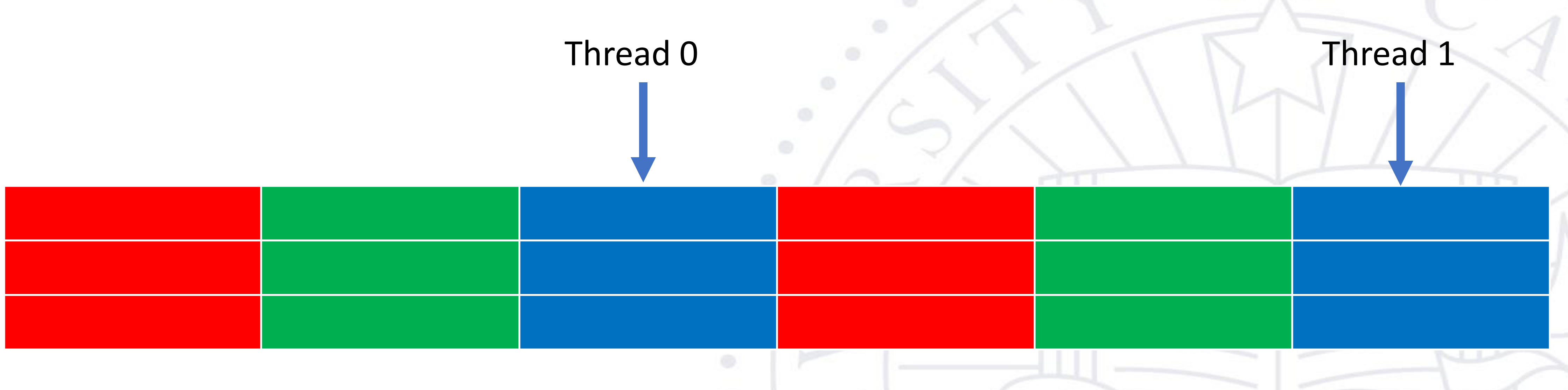
Quiz 2 – Question 2 uncoalesced

unsigned char g = rgbImage[rgbOffset + 1]; ***// green value for pixel***



Quiz 2 – Question 2 uncoalesced

unsigned char b = rgbImage[rgbOffset + 2]; ***// blue value for pixel***

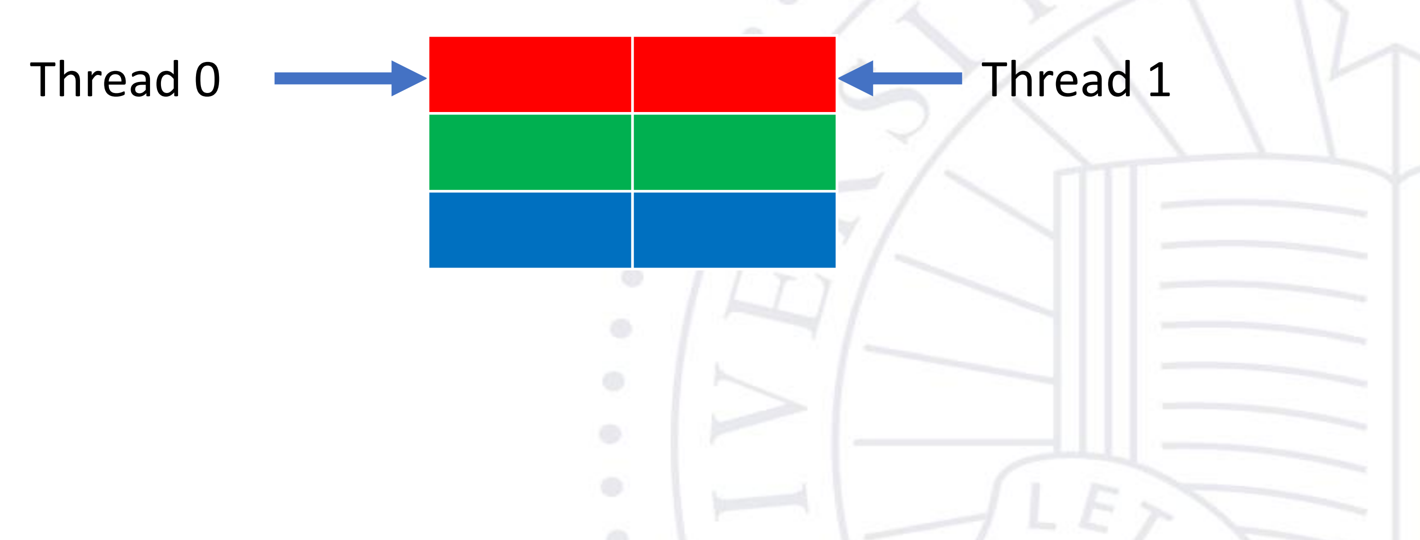


Quiz 2 – Question 2

- One way to make it coalesced is to transpose the matrix and access row by row

Quiz 2 – Question 2 coalesced

```
unsigned char r = rgbImage[rgbOffset +(width*0)]; // red value for pixel
```



Quiz 2 – Question 2 coalesced

```
unsigned char g = rgbImage[rgbOffset +(width*1)]; // green value for pixel
```

Quiz 2 – Question 2 coalesced

unsigned char b = rgbImage[rgbOffset +(width*2)]; ***// blue value for pixel***

Quiz 2 – Question 2

- One way to make it coalesced is to transpose the matrix and access row by row
- You could have stored the image in shared memory first and then kept the current access pattern, you would need to tile the loads since shared memory is limited

Quiz 2 – Question 2

- An exception: In general this does not exhibit memory coalescing However….
- This example used 1 byte characters per element [0 255]
- 32 thread warp X 3 channels per thread = 96 bytes are accessed per warp
- If our dram burst size is 128 or anything > 96 bytes then this access pattern would still be coalesced in memory
- But you would have to know the burst size which is it is not always the case

Quiz 2 – Question 3

- Implementation 2 is better for any size
- It has less warp divergence and exhibits memory coalescing
- Implementation 1 every other thread becomes inactive thus has warp divergence after the first phase
- Implementation 2 active threads are contiguous and do not have divergence until the last 5 stages (32,16,8,4,2,1)
- First five stages of no divergence only occurs if the size is 1024

Quiz 2 – Question 4

- The second one does not tile but it is more work efficient
- it does less computation O(n) compared with O(nlogn).
- It achieves this by using the reduction and then post reduction phases.
- The reduction phase computes partial sums along the vector so there is less duplication of work among threads

UNIVERSITY OF CALIFORNIA
UC RIVERSIDE

Computer Architecture

- Threads and processes
  - What they contain and how they relate in hardware and software
- Cache hierarchy
  - Understand the memory gap
  - SW leads to HW design
- Principles of spacial and temporal locality
  - How to write code to apply them
  - HW leads to SW design
- Specialization towards parallel processing
- These are foundational concepts questions will not be explicitly mentioning them but will have implied understanding

GPU Architecture

- Warps contain 32 threads and execute on a SIMD unit
- SM Cores contain multiple SIMD Units run entire Thread Blocks
- GPU Contains multiple SMs

	Scalar	Vector	Core	Card
Hardware	ALU	ALU	SIMD	SM SM
	ALU Unit	SIMD Unit	SM	GPU
Threads				
	Thread	Warp	Thread Block	Block Grid
Memory	Register File		L1 Cache	L2 / Memory
Address Space	Local per thread		Shared Memory	Global

GPU Architecture

- Hardware constraints
- Limit to number of threads and thread block per SM

Table 2. Compute Capabilities: GK180 vs GM200 vs GP100 vs GV100

GPU	Kepler GK180	Maxwell GM200	Pascal GP100	Volta GV100
Compute Capability	3.5	5.2	6.0	7.0
Threads / Warp	32	32	32	32
Max Warps / SM	64	64	64	64
Max Threads / SM	2048	2048	2048	2048
Max Thread Blocks / SM	16	32	32	32
Max 32-bit Registers / SM	65536	65536	65536	65536
Max Registers / Block	65536	32768	65536	65536
Max Registers / Thread	255	255	255	255[1]
Max Thread Block Size	1024	1024	1024	1024
FP32 Cores / SM	192	128	64	64
Ratio of SM Registers to FP32 Cores	341	512	1024	1024
Shared Memory Size / SM	16 KB/32 KB/ 48 KB	96 KB	64 KB	Configurable up to 96 KB

[1] The per-thread program counter (PC) that forms part of the improved SIMT model typically requires two of the register slots per thread.

GPU Architecture

- Hardware constraints examples
- An SM is fully occupied if it is running the maximum number of threads
- 2 blocks with 1024 threads – Fully occupied
- 32 blocks with 32 threads – not fully occupied
- Typically you want the number of threads per block to be divisible by 32 and have at least 64 threads
- Multidimensional blocks get linearlized
- Block size of (16,16,4) = 16*16*4 =1024 threads

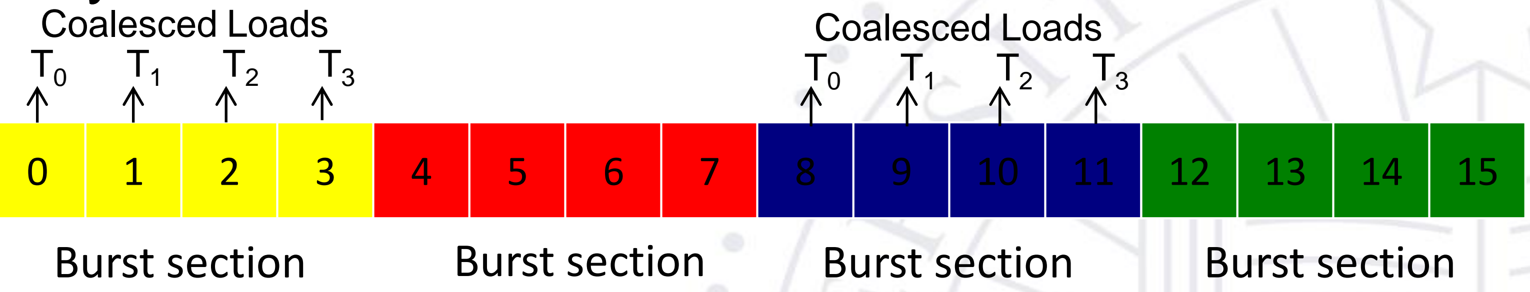
Max warps / SM	64
Max Threads / SM	2048
Max Thread Blocks / SM	32
Max Thread Block Size	1024

CUDA Programming

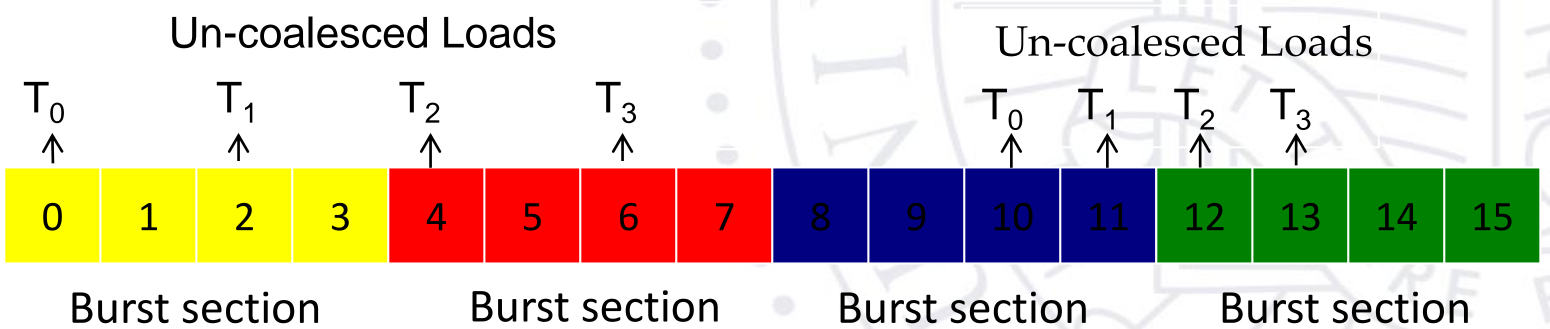
- Allocate, Copy to Device, Launch, Copy to Host
  - Cudamemcopy(dest,src,size,direction)
  - globalFunction<<<gridDim,BlockDim>>>(args)
- Allocate and copy data only pointed to by pointers
- Block and Grid size are 3 Dimensional
- Threads are assigned a Thread id and Block id in each dimension
  - Determine proper block and grid size for any input size
  - How to assign data with thread and block ids e.g...
  - Row = blockIdx.y*blockDim.y + threadIdx.y;
  - Col = blockIdx.x*blockDim.x + threadIdx.x;

Memory coalescing

- When all threads of a warp execute a load instruction, if all accessed locations are contiguous, only one DRAM request will be made and the access is fully coalesced.



- When the accessed locations spread across burst section boundaries Coalescing fails and Multiple DRAM requests are made

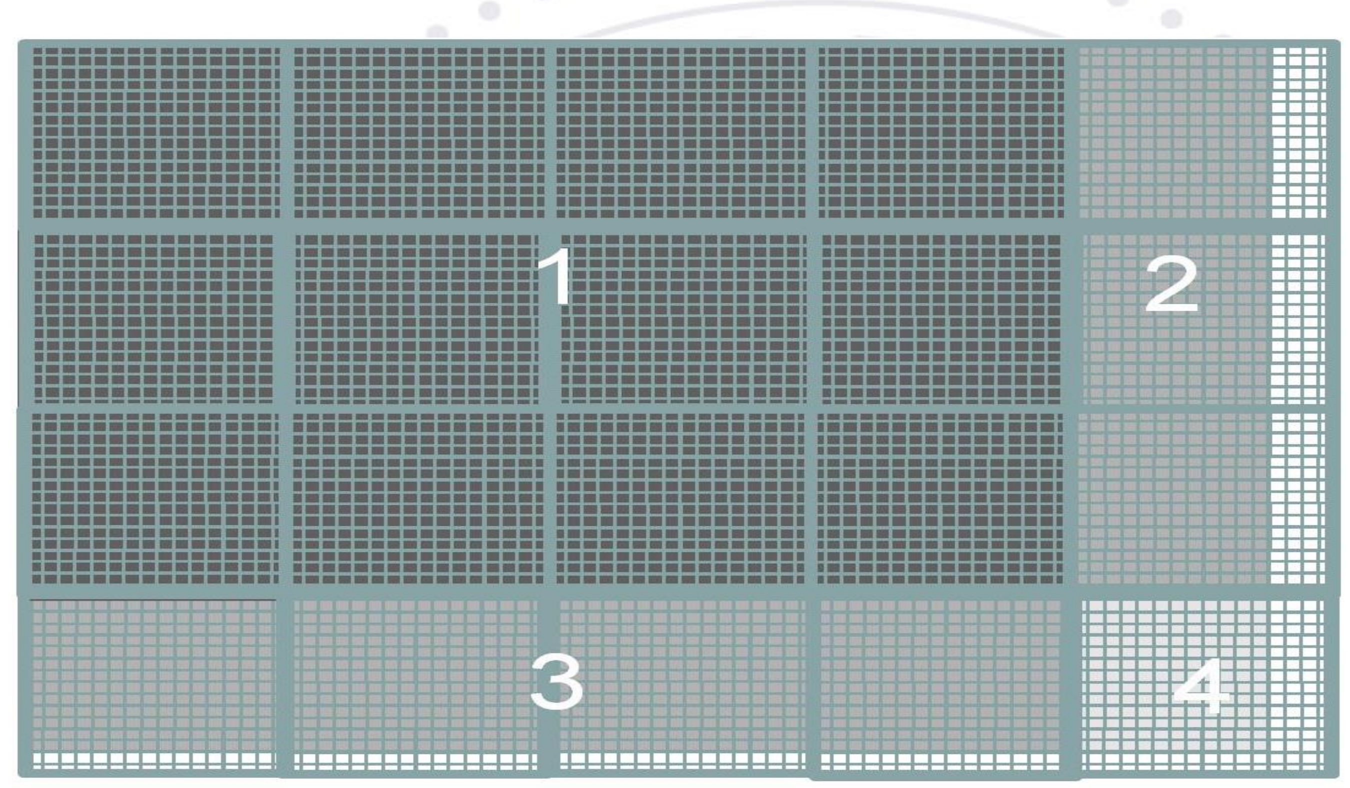


Memory coalescing

- Be able to spot and modify code to address memory coalescing concerns
- This affect thread access patterns
- Loads across threads access memory contiguously
- Threads read across a row and access down a column
- Or load into shared memory if your access pattern cannot be easily altered

Warp Divergence

- Divergence only occurs when threads within a warp go through different control paths
- 1) all threads are active
- 2) All warps have divergence
- 3) Some threads are inactive but no warp divergence
- 4) Some warps have divergence



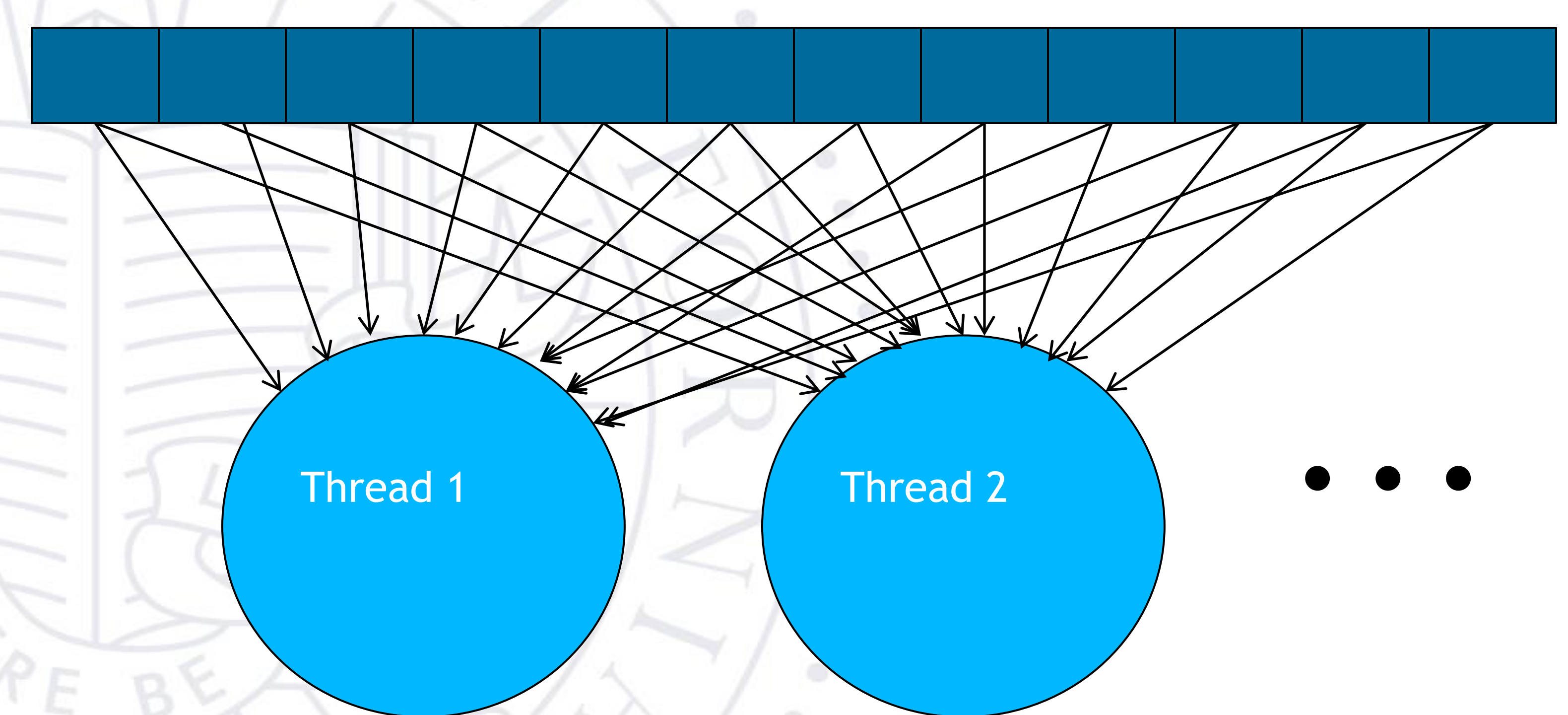
Warp Divergence

- Be able to calculate the number of warps that exhibit divergence for a particular input and block size
- Spot and modify code to reduce the amount of divergence
  - Pad outer bounds with 0 and get rid of any control instructions
  - Resize block or change thread access pattern to land on warp boundaries
  - Compact active threads to contiguous warps (reduction implementation)

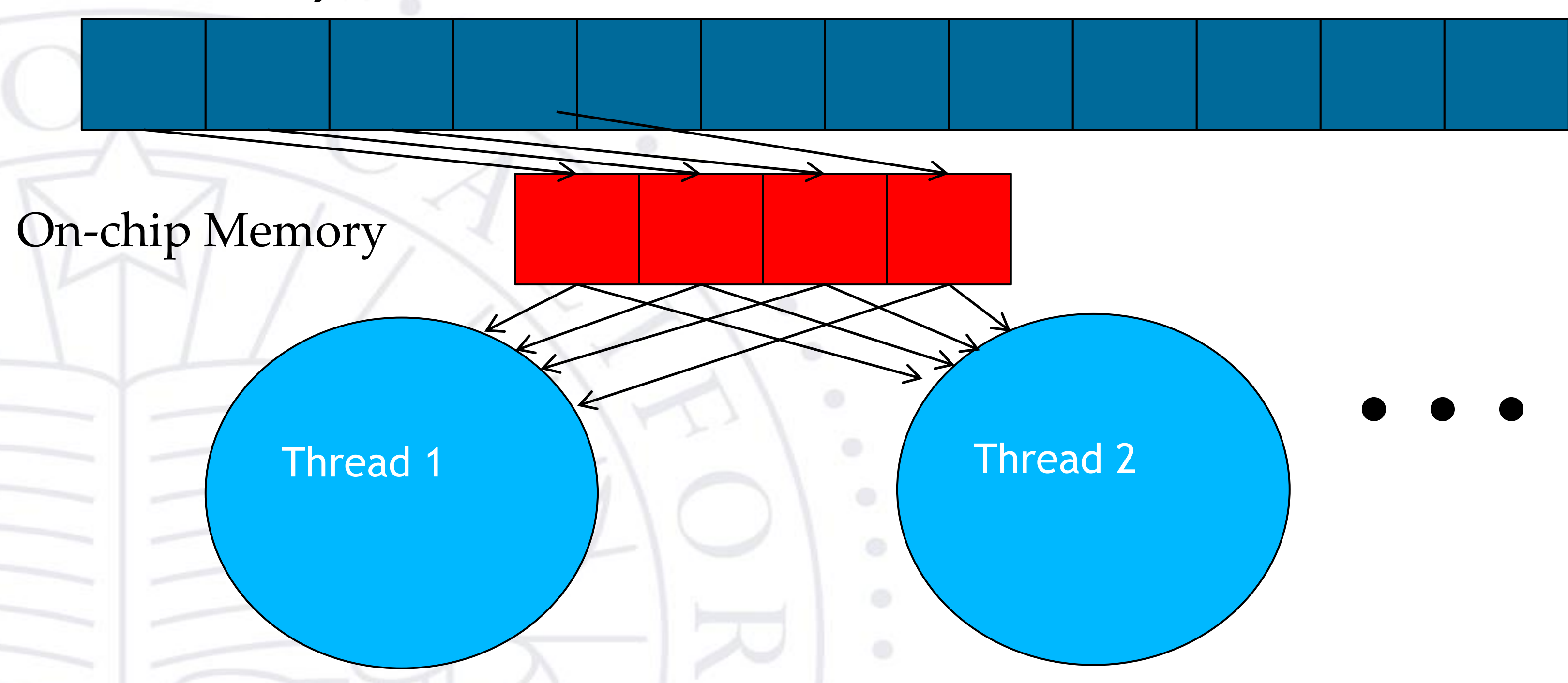
Shared memory

Accessing memory is expensive, reduce the number of global memory loads

Global Memory



Shared Memory



Divide the global memory content into tiles

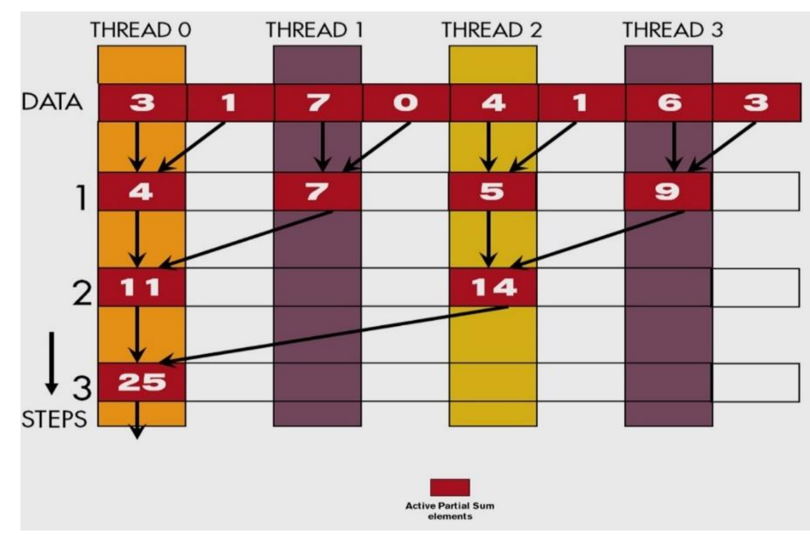
Focus the computation of threads on one or a small number of tiles at each point in time

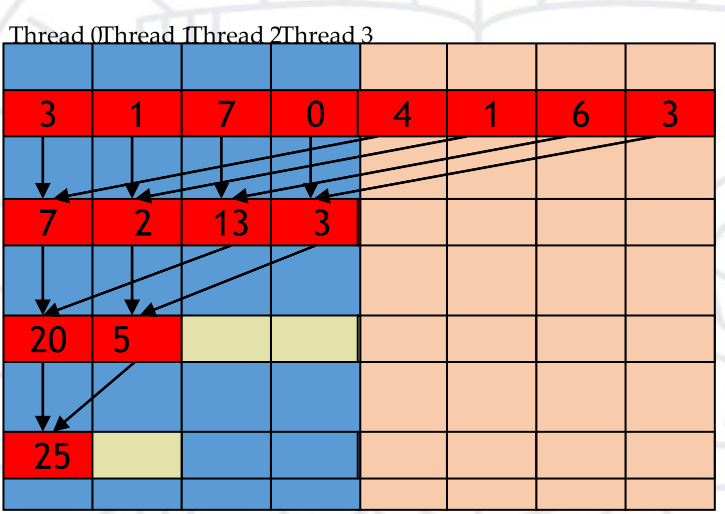
Shared Memory

- Declare with __Shared__ var[size]
- Load into shared var then read from it
- Shared memory is only useful if you access it multiple times
- How to use it with tiling

Reduction

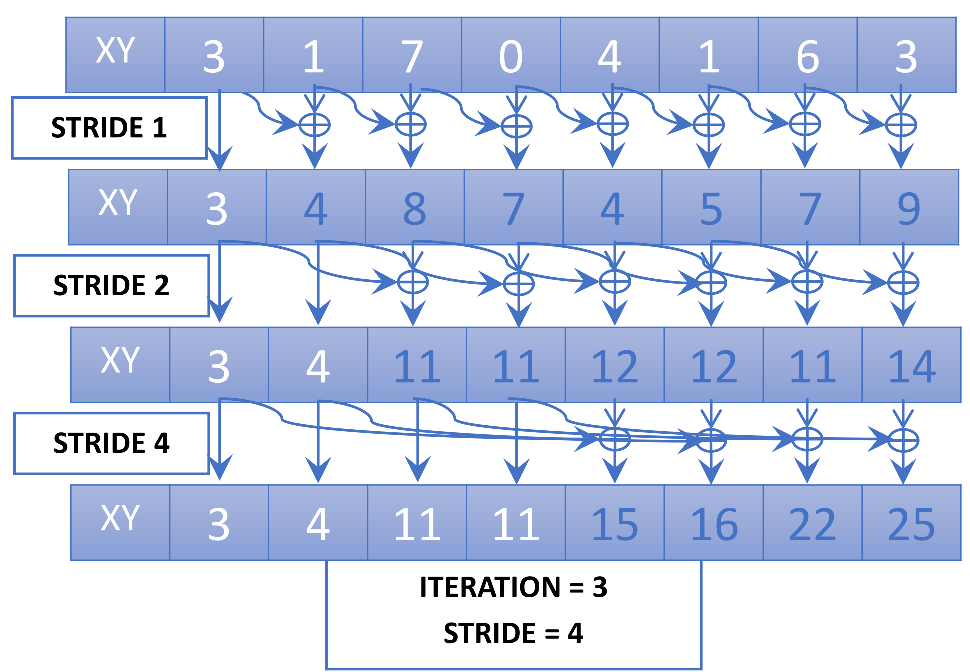
- Parallel reduction uses tree algorithm for O(logn)
- Two implementations
  - Understand the difference in implementation and performance
- Understand as an example of warp divergence, memory coalescing, and thread synchronization

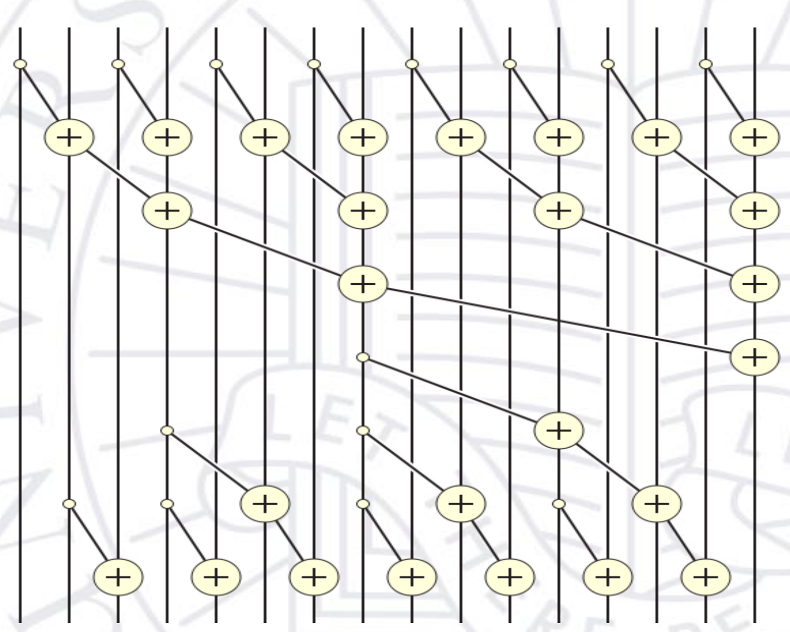




Scan

- Parallel scan either strided array or tree algorithm
- Two implementations
  - Understand the difference in implementation and performance
- Understand as an example of work efficiency and thread synchornization





Tiled Matrix Multiplication

- Great example of tiling algorithm, use of shared memory, and thread synchronization
- Relation between tile size and block size
- Number of tiled phases for any height and width of matrix
- 2D Thread and block ids