CUDA Optimizations

WS 2014-15 Intelligent Robotics Seminar

Table of content

Table of content

Background Information

- Why GPUs?
- Your PC with GPU
- Understanding SM and memory hierarchies
- Understanding CUDA kernel launch
- Questions?

Background Information

It’s all about real time

- Motion compliance < 1 ms
- Vision (30fps) < 33 ms
- Vision (60fps) < 16 ms

Neural Networks

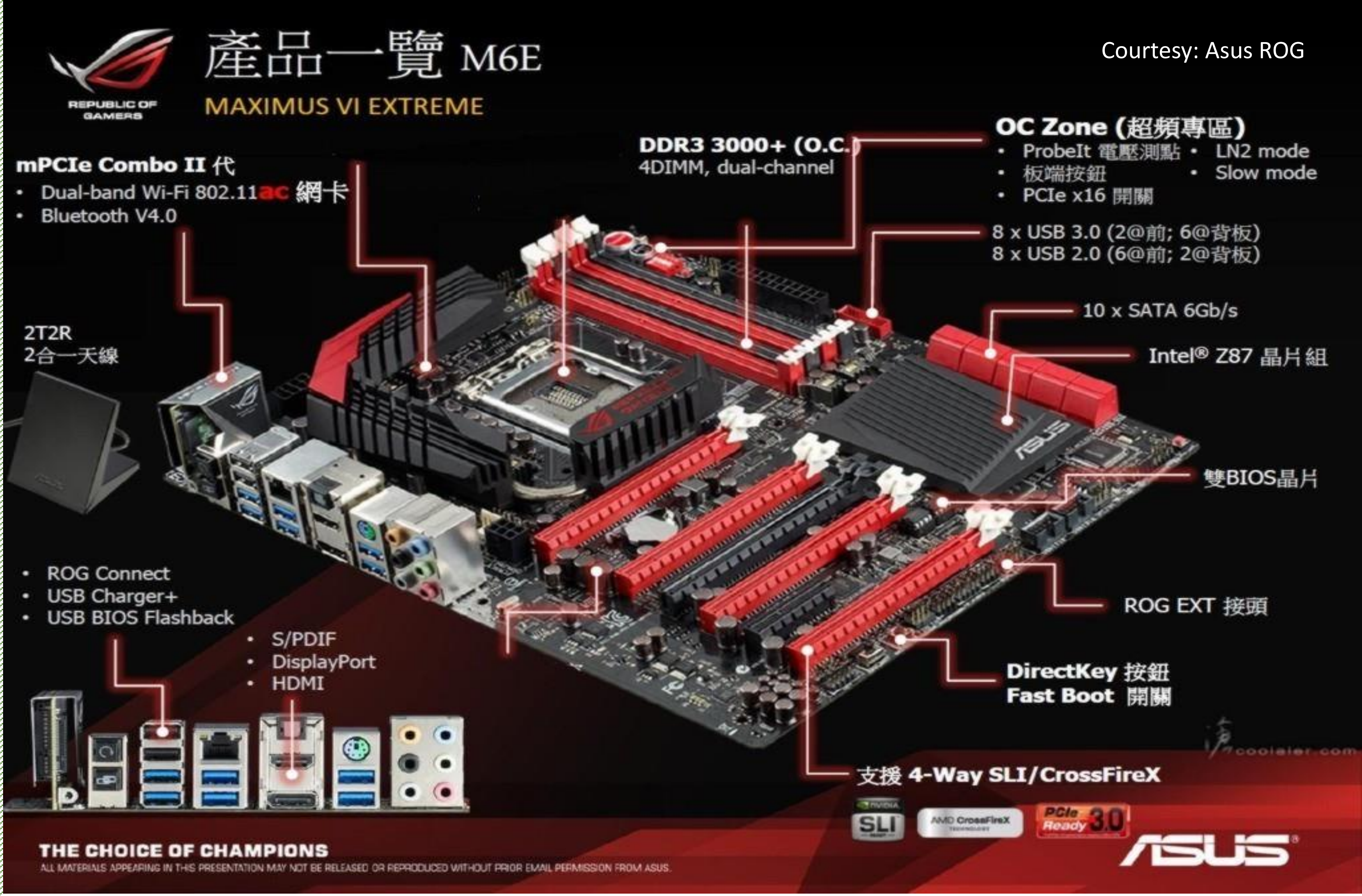
- Neuron within a neural network computes its own activation based on local information
- Learning algorithms continuously adapt the strength of connections between neurons

pre-processing

- accelerates some of the pre-processing required (e.g. vision processing)

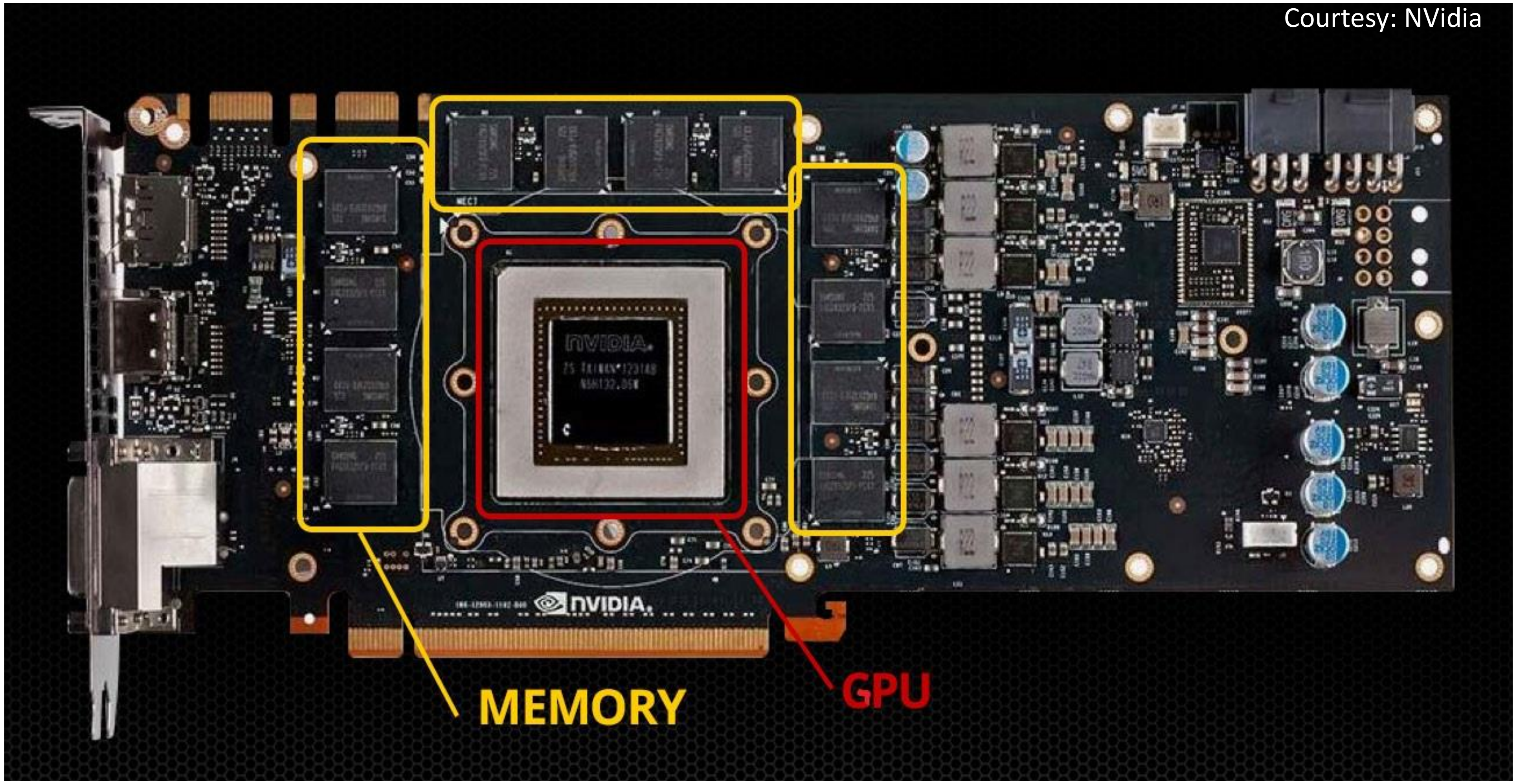
Background Information

- Why GPUs?
- Your PC with GPU
- Understanding SM and memory hierarchies
- Understanding CUDA kernel launch
- Questions?



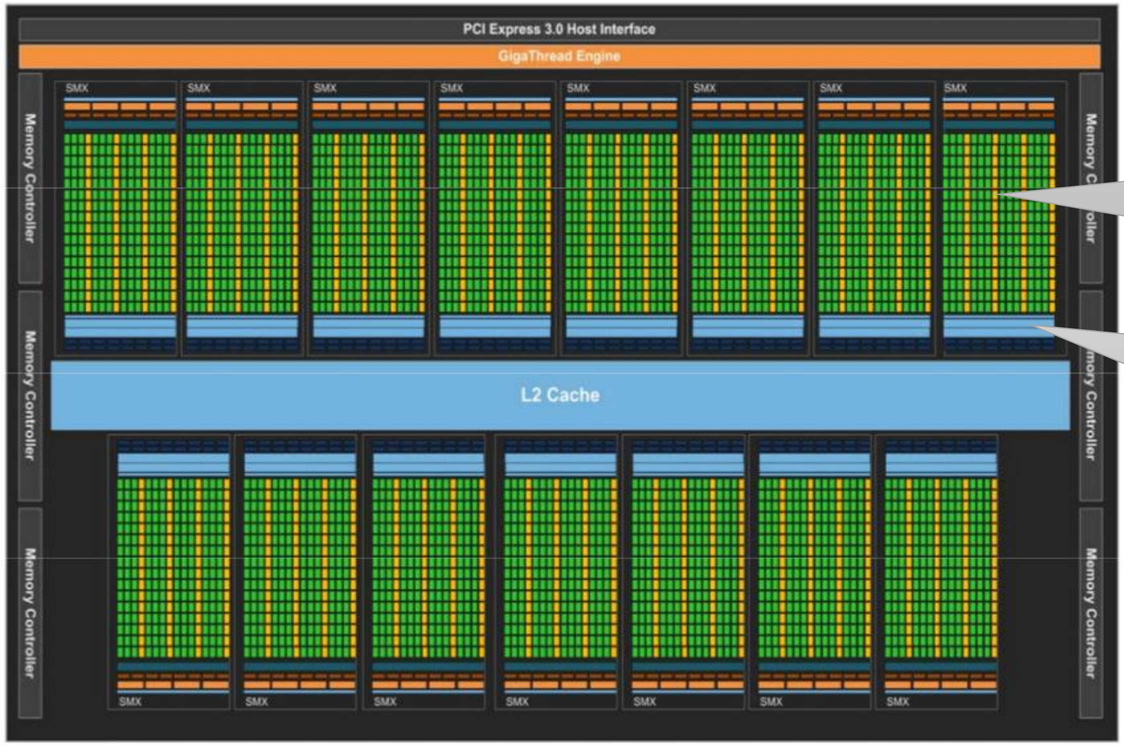
Courtesy: Asus ROG

Background Information



Global memory (off chip DDR5 RAM)

- Off chip memory
- Constant and texture memory also allocated here



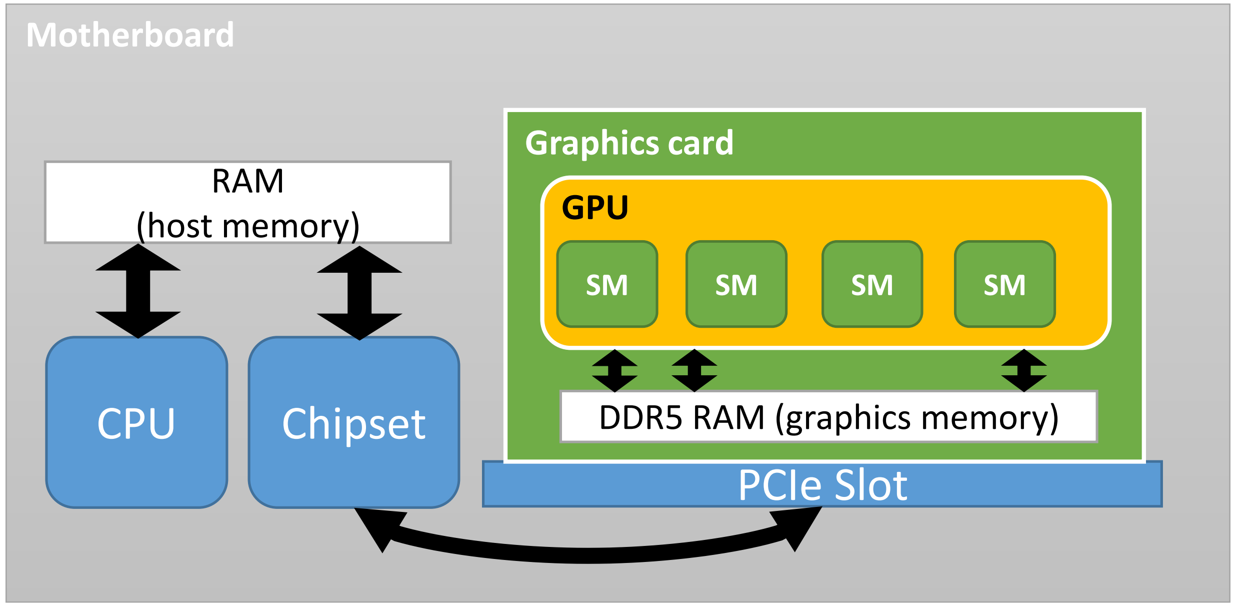
- SM (streamed multi-processor)
- Blocks of threads are scheduled on SM (e.g. group of 512 threads)

- Shared memory which can be shared between threads in block

Global memory (off chip DDR5 RAM)

Background Information

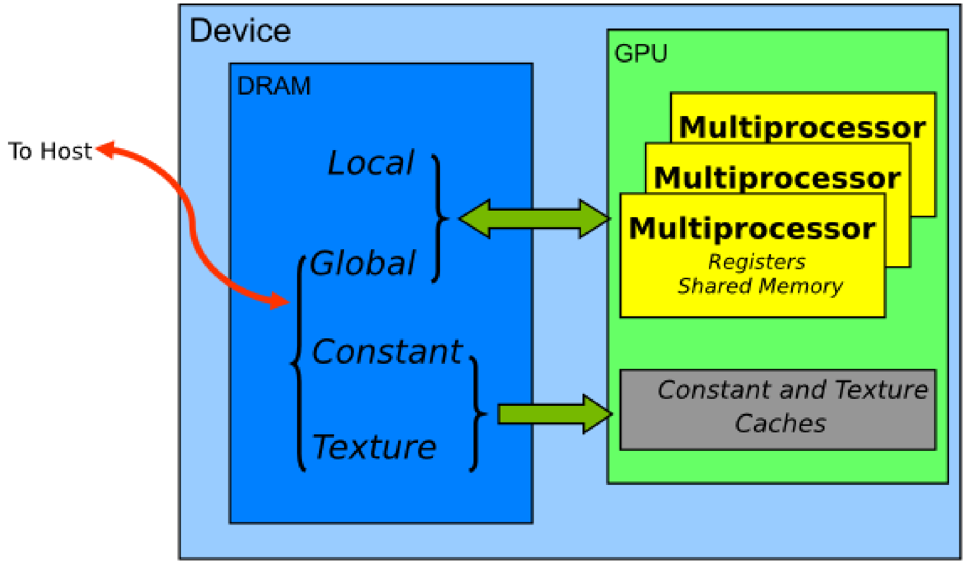
- Why GPUs?
- Your PC with GPU
- Understanding SM and memory hierarchies
- Understanding CUDA kernel launch
- Questions?



Fastest memory in rough order

On chip
- Registers
- Shared

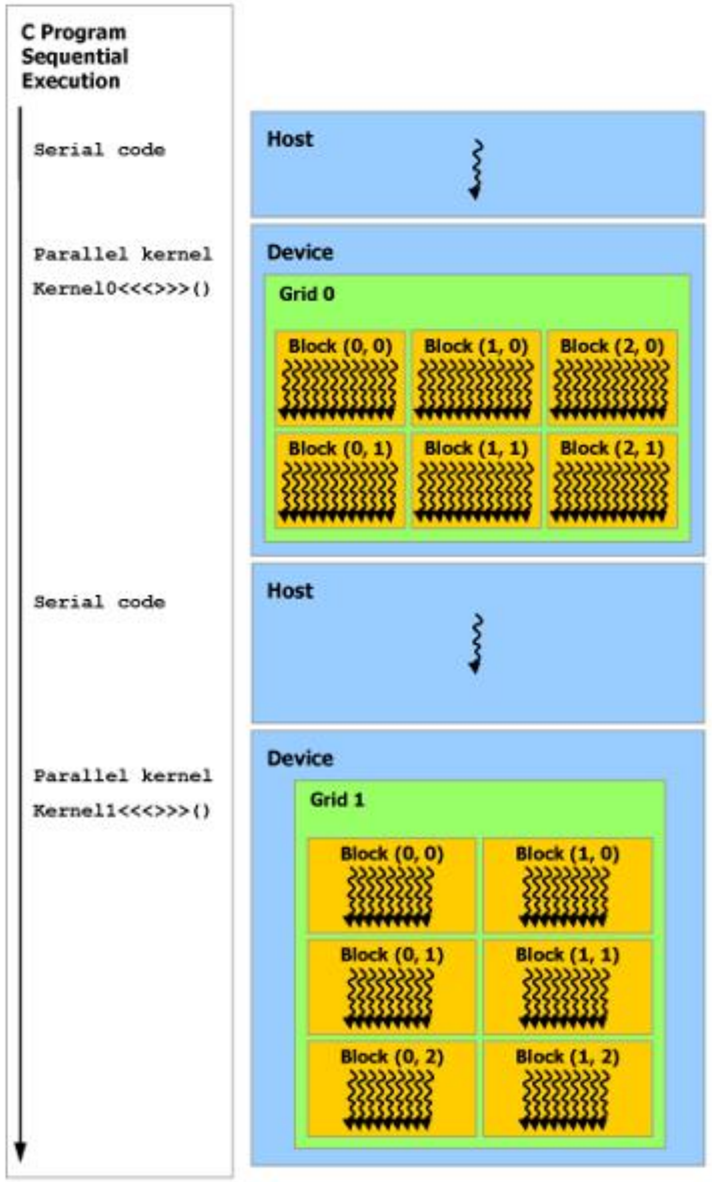
Off chip
- Constant
- Texture
- Global



Memory	Location on/off chip	Cached	Access	Scope	Lifetime
Register	On	n/a	R/W	1 thread	Thread
Local	Off	†	R/W	1 thread	Thread
Shared	On	n/a	R/W	All threads in block	Block
Global	Off	†	R/W	All threads + host	Host allocation
Constant	Off	Yes	R	All threads + host	Host allocation
Texture	Off	Yes	R	All threads + host	Host allocation

† Cached only on devices of compute capability 2.x

Background Information



Which of these is on-chip memory for GPU?

- Host memory (RAM)
- ✓ Registers
- ✓ Shared memory
- Global memory

Can threads in different blocks access same shared memory?

- Yes
- ✓ No

Order memories based on speed

5	Host memory (RAM)
1	Registers
4	Global memory (GPU memory)
3	Constant and texture memory
2	Shared memory

Which of these memories are persistent?

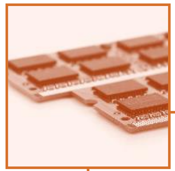
- Registers
- ✓ Global memory (GPU memory)
- ✓ Constant and texture memory
- Shared memory

Except for constant and texture memory, all other memories are R/W

- ✓ Yes
- No

Table of content

Categorized optimization strategies



Memory optimizations

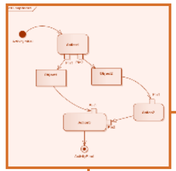
- Data Transfer Between Host and Device
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- Device Memory Spaces
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

Execution Configuration optimizations

- Occupancy
- Concurrent Kernel Execution
- Hiding Register Dependencies
- Thread and Block Heuristics
- Effects of Shared Memory

Instruction optimization

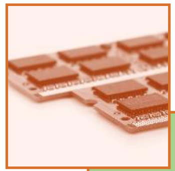
- Arithmetic Instructions
- Memory Instructions



Control flow

- Branching and Divergence
- Branch Predication
- Loop Counters Signed vs. Unsigned
- Synchronizing Divergent Threads in a Loop

Categorized optimization strategies



Memory optimizations

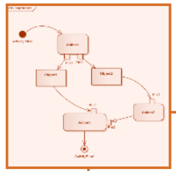
- Data Transfer Between Host and Device
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- Device Memory Spaces
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

Execution Configuration optimizations

- Occupancy
- Concurrent Kernel Execution
- Hiding Register Dependencies
- Thread and Block Heuristics
- Effects of Shared Memory

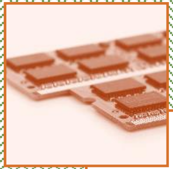
Instruction optimization

- Arithmetic Instructions
- Memory Instructions



Control flow

- Branching and Divergence
- Branch Predication
- Loop Counters Signed vs. Unsigned
- Synchronizing Divergent Threads in a Loop



Memory optimizations

- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

Goal for Memory optimizations

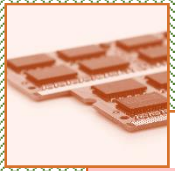
- maximize the use of the hardware by maximizing bandwidth

maximizing bandwidth

- using as much fast memory and as little slow-access memory as possible

What follows next

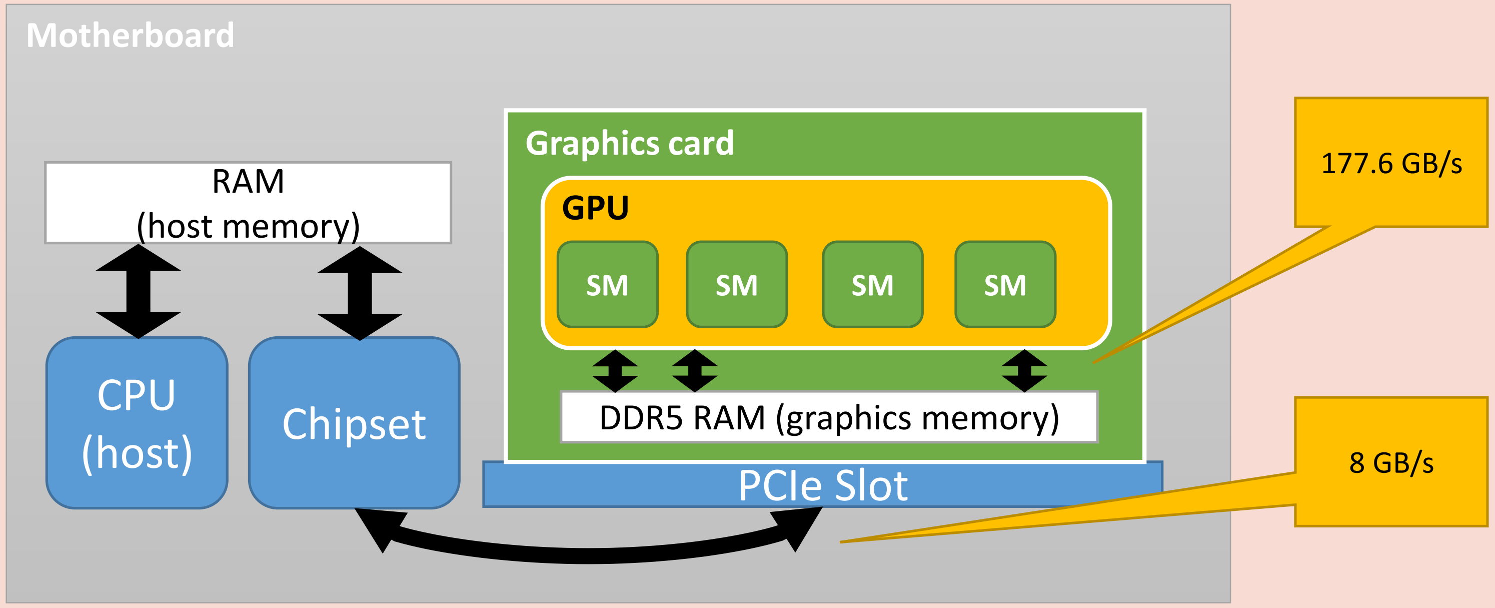
- discuss the various kinds of memory on the host and device and how best to set up data items to use the memory effectively



Memory optimizations

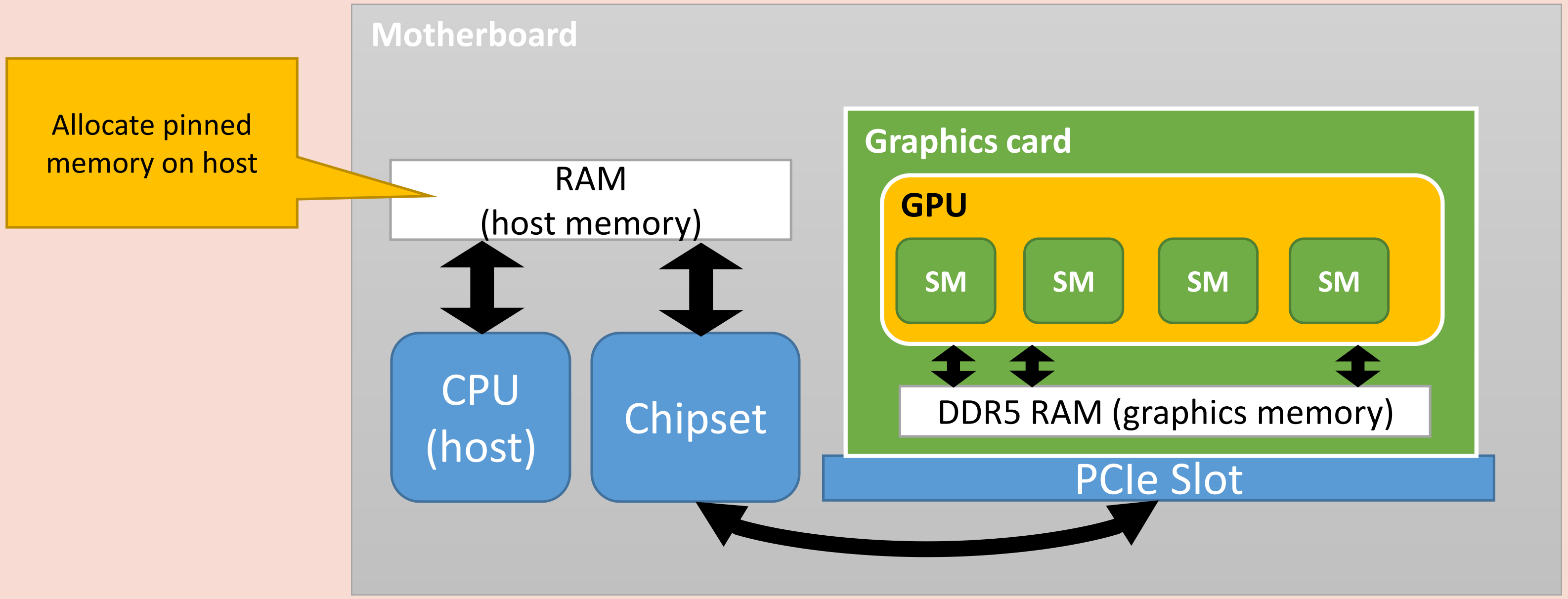
- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

Why data transfer between host and device must be minimized

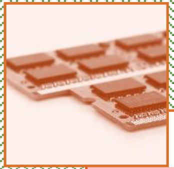


- 177.6 GB/s > 8 GB/s
- Its fine even if we run kernels on the GPU that do not demonstrate any speedup

pinned memory improves transfer between host and device



- Page-locked or pinned memory transfers attain the highest bandwidth between the host and the device
- can reduce overall system performance (since it is scarce resource)
- Pinning memory is heavy weight operation



Lets assume that you are doing some processing on an image.

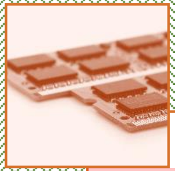
In which scenarios will you use pinned memory?

✗ •A] You have very limited host memory (RAM)

✗ •B] The image processing algorithm running on GPU has many steps to be performed on image

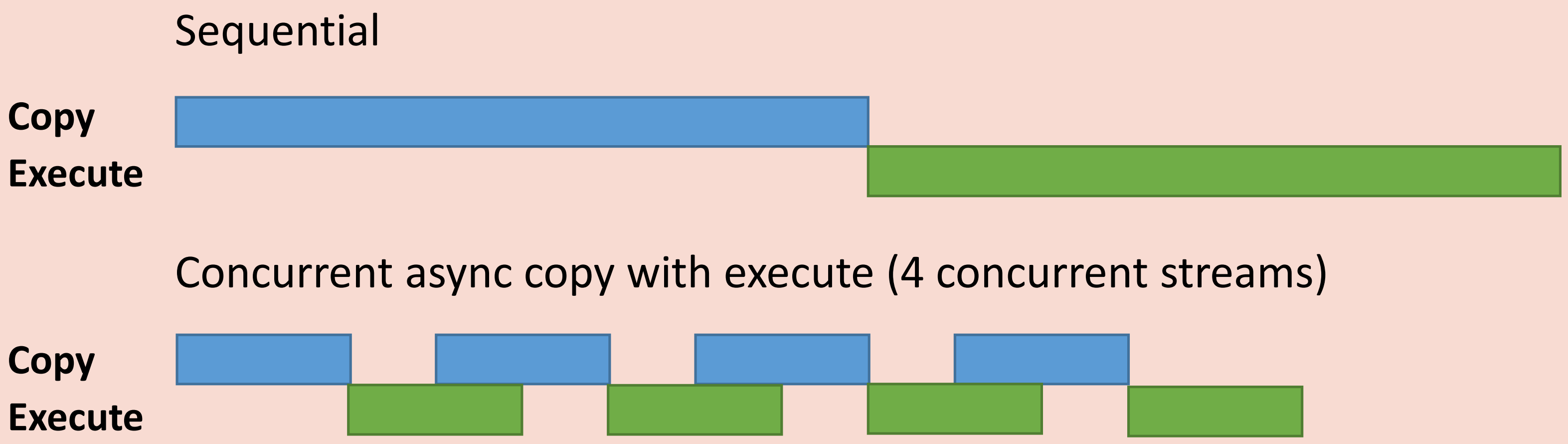
✓ •C] Your application demands to have processed image always available with host CPU

Memory optimizations



- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

Staged concurrent copy and execute

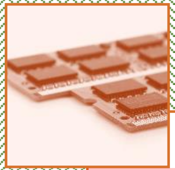


Memory optimizations

- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

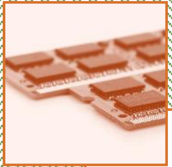
Memory optimizations

- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers



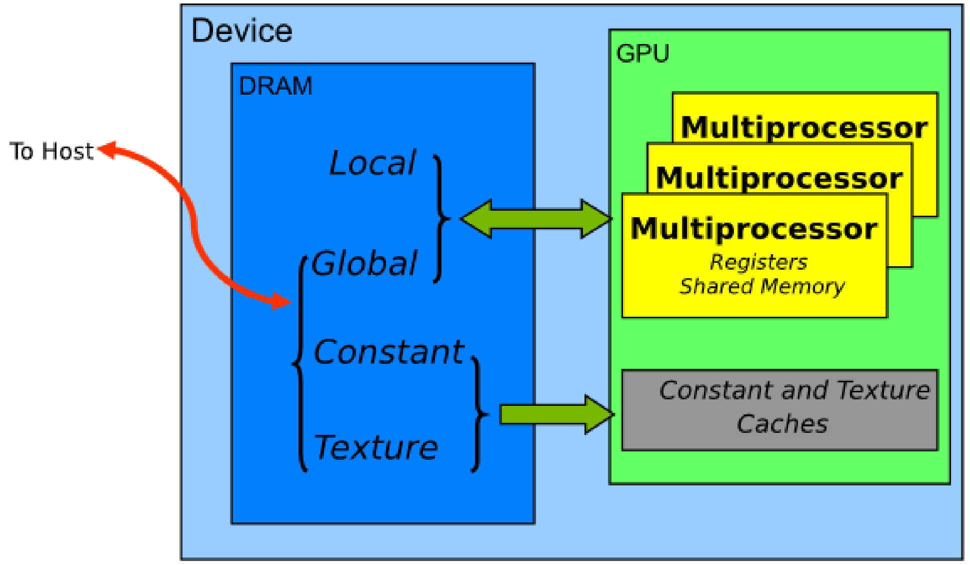
Unified Virtual Addressing

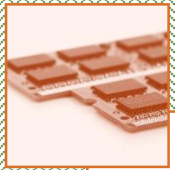
- Internally manages the address spaces and do necessary memory transfers
- Coding simplicity and rapid prototyping
- Future compatibility



Fastest memory in rough order

On chip	• Registers • Shared
Off chip	• Constant • Texture • Global





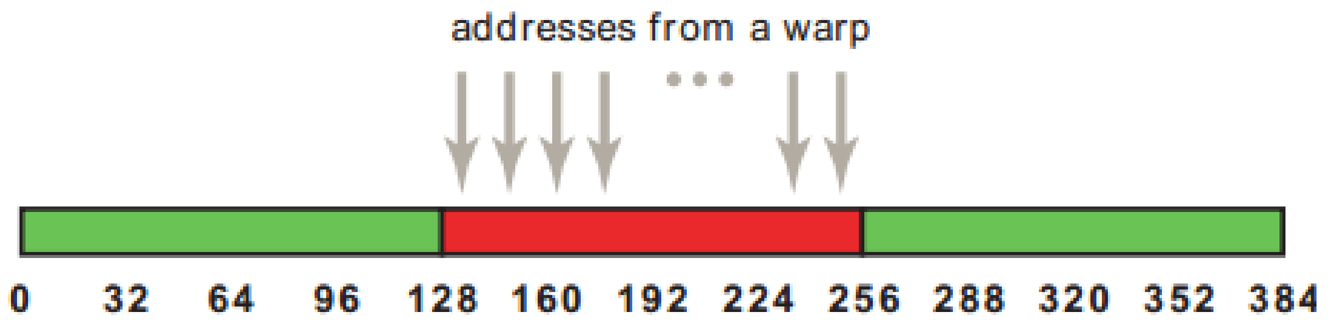
Coalesced Access to Global Memory

- Memory loads and store by threads in warps are coalesced
- RAM are designed for batch access and we can take advantage of that in programming
- We will see what happens with coalesced access to global memory when
- 1] we change offset
- 2] we change stride

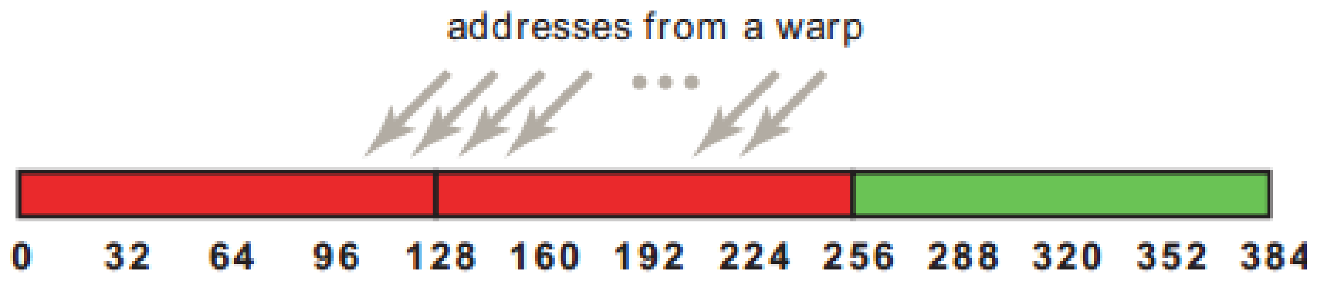
Memory optimizations

- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

1] With different offset

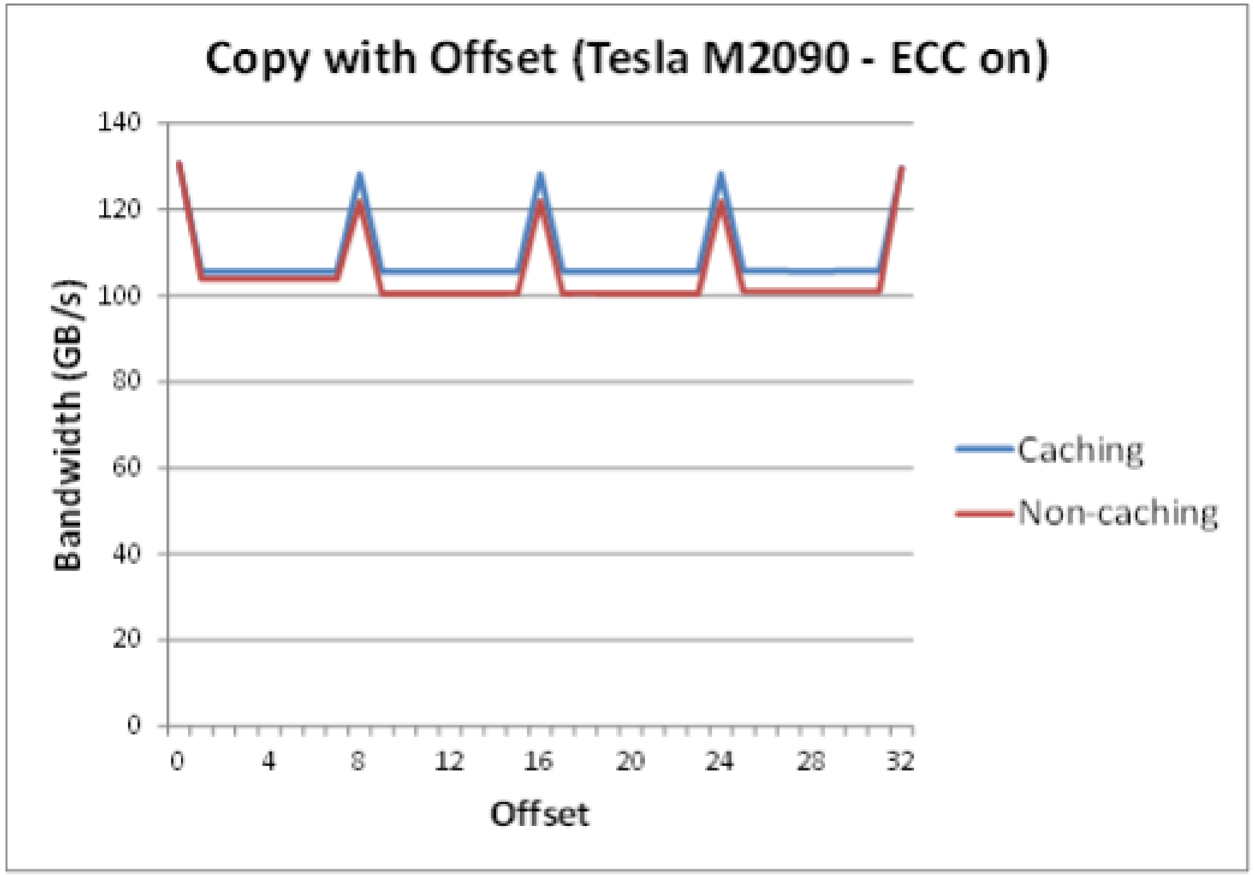


Coalesced access - all threads access one cache line



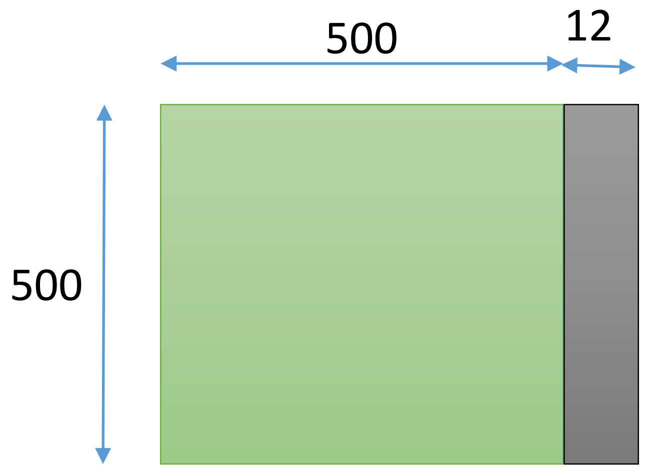
Unaligned sequential addresses that fit into two 128-byte L1-cache lines

1] With different offset

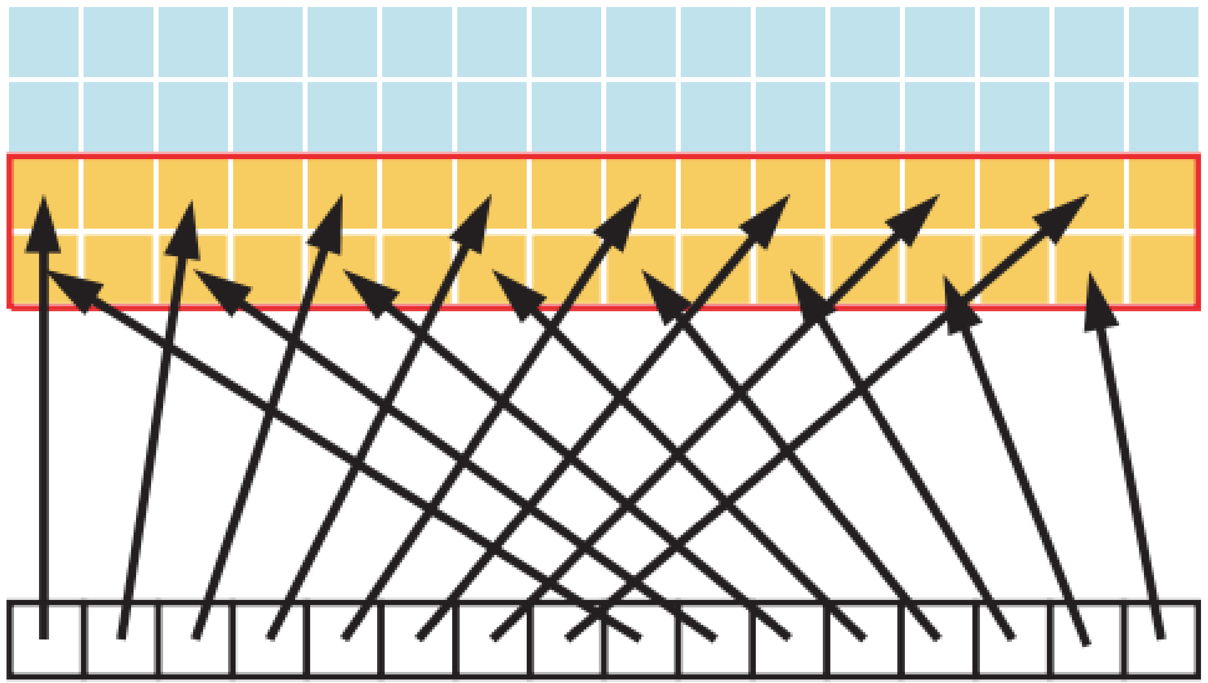


Assume we are working on an float image of size 500 X 500.

Will it be a problem? If yes what is the solution?

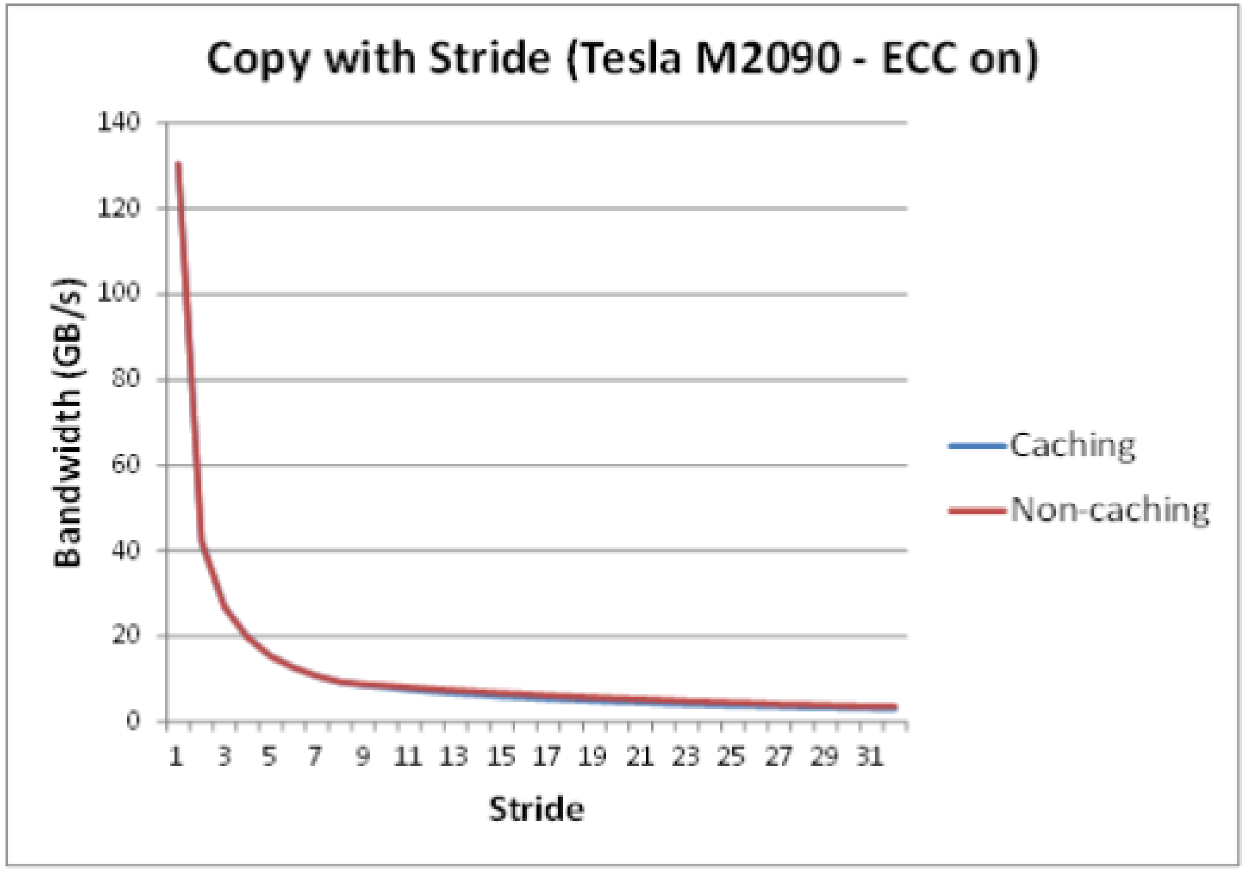


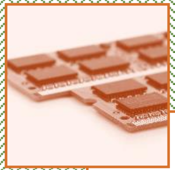
2] With different stride



Adjacent threads accessing memory with a stride of 2

2] With different stride





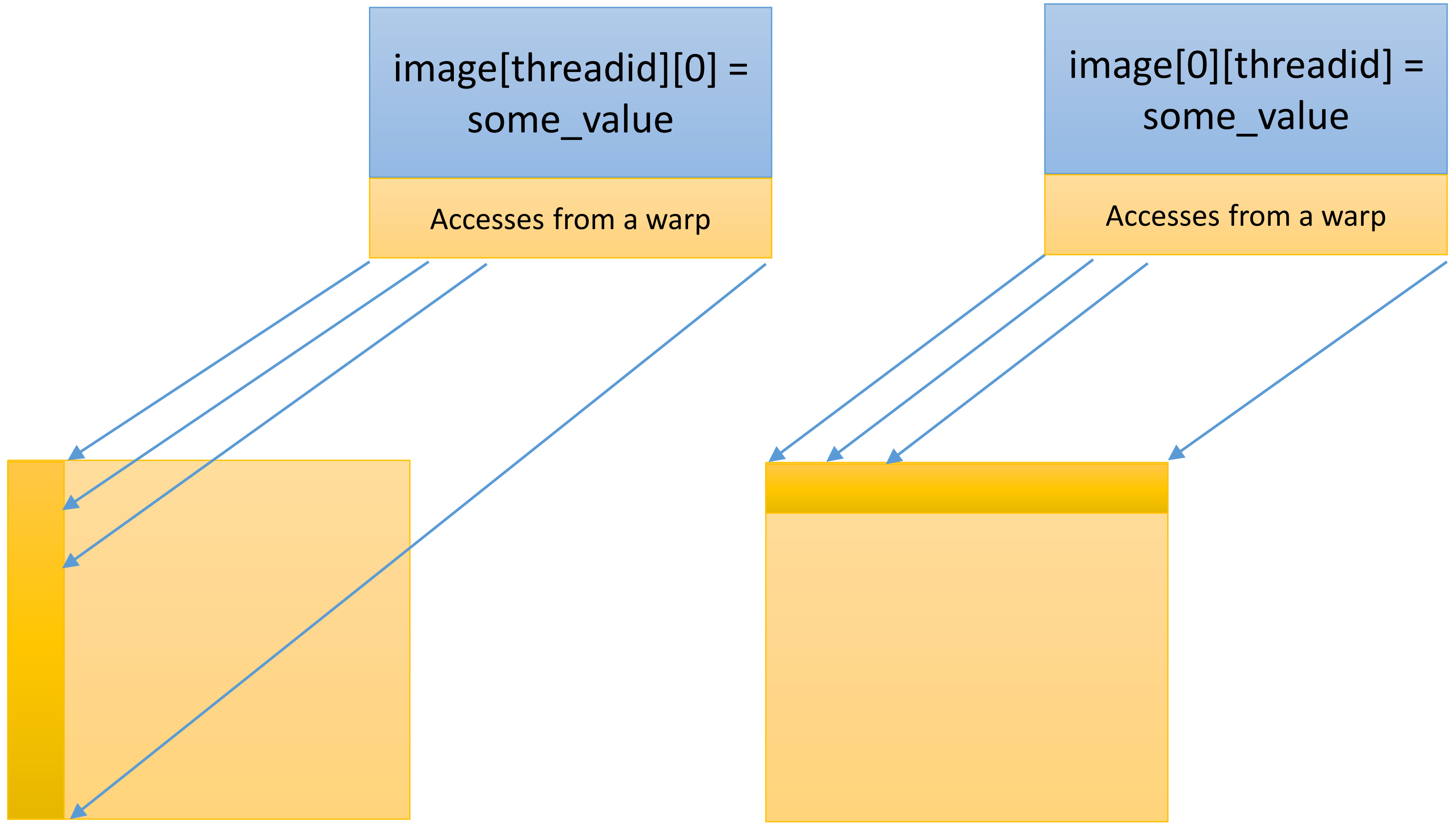
Memory optimizations

- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

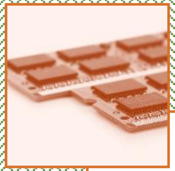
Assume we are working on an float image of size 512 X 512.

Will the below access pattern pose problem?

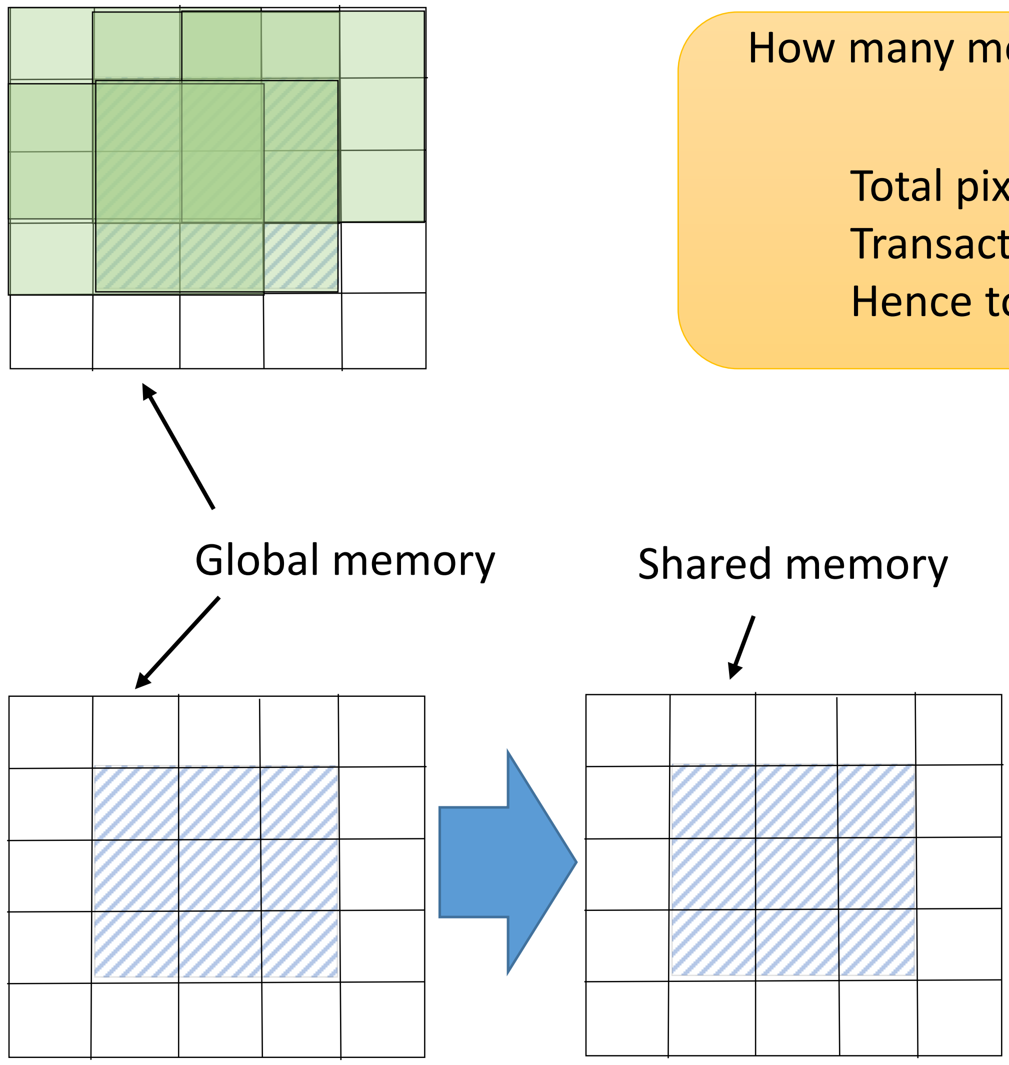
What is the stride number in this case?



image[threadid][0] = some_value
Accesses from a warp
image[0][threadid] = some_value
Accesses from a warp



- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

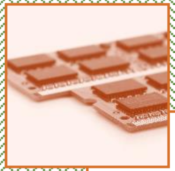


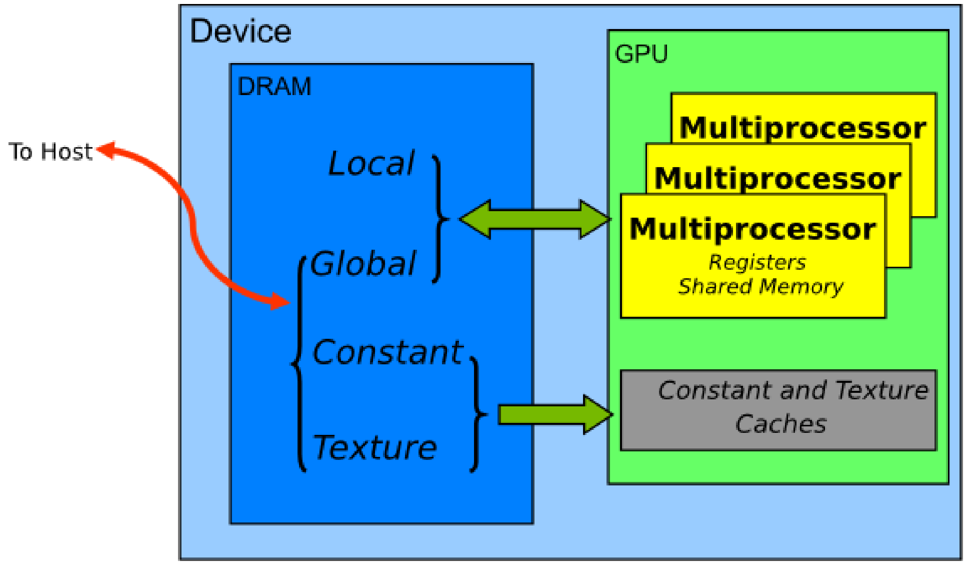
How many memory global transactions will be needed?

Total pixels to be calculated = 9
Transactions per pixel = 9
Hence total transactions = 9*9 = 81

How many global transactions with shared memory?

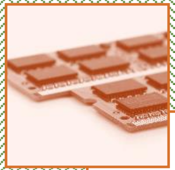
25





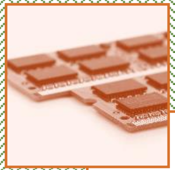
Local memory

- SM have limited register space
- Automatic variables are allocated on registers (e.g. local variables)
- If registers memory is not enough then the local memory is used. This is called **register spilling**.
- Local memory resides on global memory and hence is slow
- After compilation nvcc compiler can report local memory usage. You must try to avoid it if possible.



Texture memory

- read-only texture memory space is cached
- texture cache is optimized for 2D spatial locality
- In some cases advantageous alternative to reading device memory from global or constant memory
- Hardware provides other capabilities when textures are fetched using tex1D(), tex2D(), or tex3D() rather than tex1Dfetch()
- Filtering
- Normalized texture coordinates
- Automatic handling of boundary cases

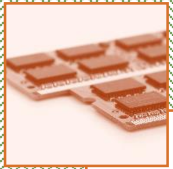


Memory optimizations

Constant memory

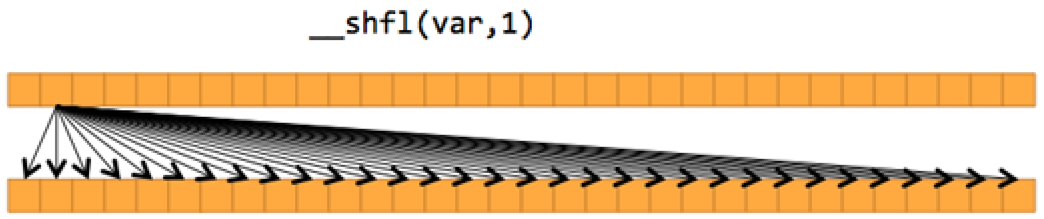
- There is a total of 64 KB constant memory on a device
- constant memory space is cached
- In some cases advantageous alternative to reading device memory from global or constant memory
- the constant cache is best when threads in the same warp accesses only a few distinct locations
- If all threads of a warp access the same location, then constant memory can be as fast as a register access

Memory optimizations



Registers

- Registers is the fastest memory space
- CUDA provides capability to uses small constant arrays
- hardware instruction support for sharing registers between threads in warp

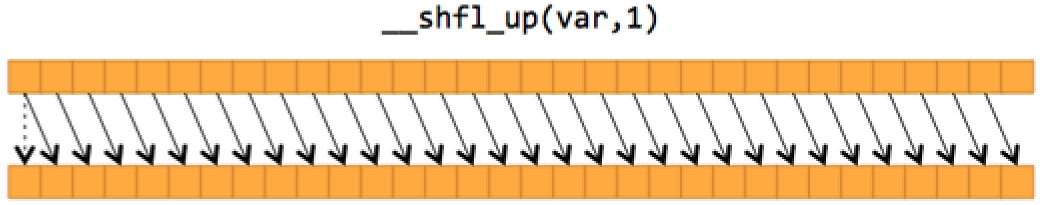


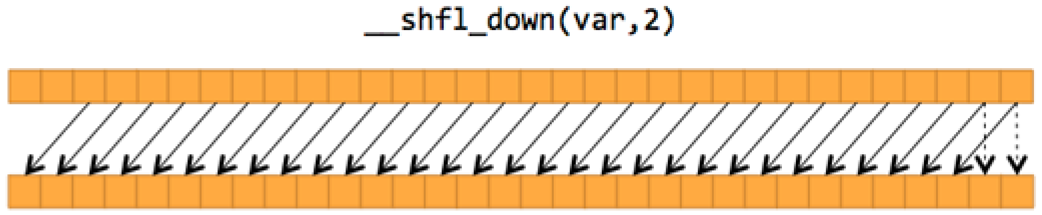
Shuffle instruction with constant srcLane broadcasts the value in a register from one thread to all threads in a warp

Memory optimizations

- **Data Transfer Between Host and Device**
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- **Device Memory Spaces**
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

Shuffle up and down instructions illustrated

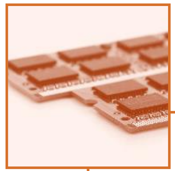




advantage of the shuffle instruction for the moving average filter algorithm

$$y[i] = \frac{\sum_{j=i-2}^{j=i+2} x[j]}{5}$$

Categorized optimization strategies



Memory optimizations

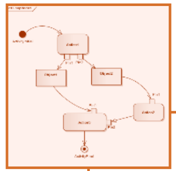
- Data Transfer Between Host and Device
  - Pinned Memory
  - Asynchronous and Overlapping Transfers with Computation
  - Unified Virtual Addressing
- Device Memory Spaces
  - Coalesced Access to Global Memory
  - Shared Memory
  - Local Memory
  - Texture Memory
  - Constant Memory
  - Registers

Execution Configuration optimizations

- Occupancy
- Concurrent Kernel Execution
- Hiding Register Dependencies
- Thread and Block Heuristics
- Effects of Shared Memory

Instruction optimization

- Arithmetic Instructions
- Memory Instructions



Control flow

- Branching and Divergence
- Branch Predication
- Loop Counters Signed vs. Unsigned
- Synchronizing Divergent Threads in a Loop

Questions?

END