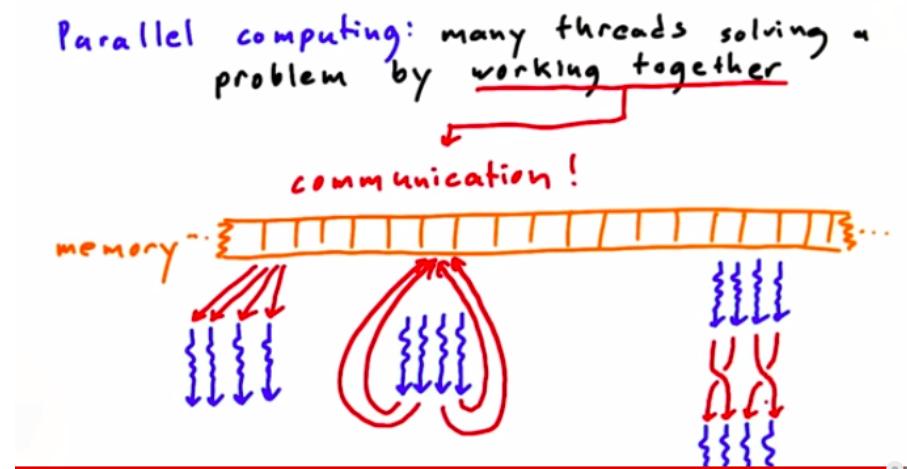


CS344 Introduction to Parallel Programming

Lesson 2: GPU Hardware and Parallel Communication Patterns

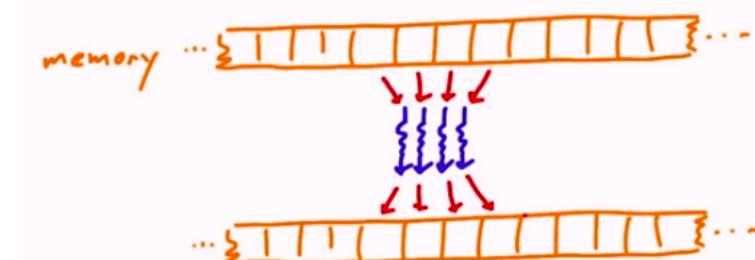
L2-2.2-Communication Patterns



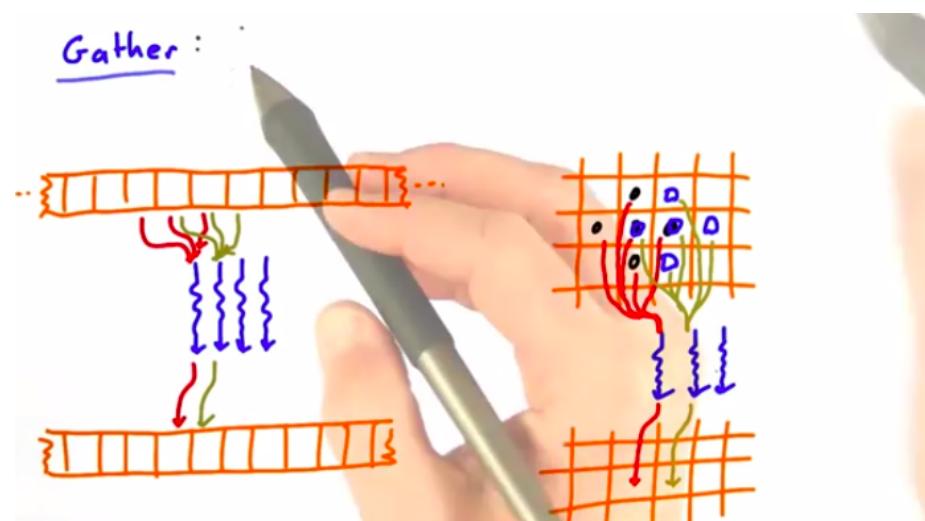
L2-2.3-Map and Gather

Parallel Communication Patterns

Map: Tasks read from and write to specific data elements

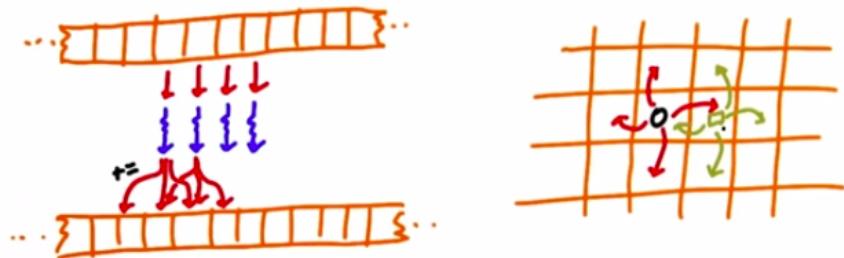


Gather:



L2-2.4- Scatter Quiz

Scatter: tasks compute where to write output



Quiz: Given a list of basketball players

- name
- height
- rank in height (tallest, 2nd tallest, ...)

Task: write each player's record into its location in a sorted list.



Is this:

- Map?
- Gather?
- Scatter?

L2-2.5-Stencil-Quiz

Stencil: tasks read input from a fixed neighborhood in an array. Data Reuse!

stencil

Quiz: How many times will a given input value be read when applying each stencil?



2D von Neumann

—



2D Moore

—



3D von Neumann

—

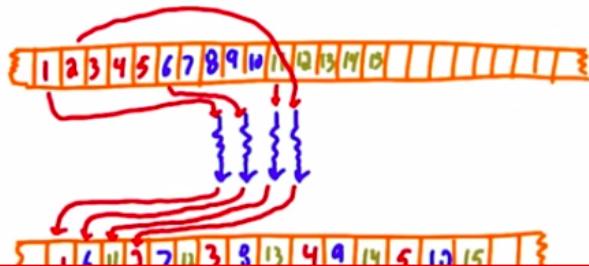
L2-2.6-Transpose Part 1

Transpose

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15



1	6	11
2	7	12
3	8	13
4	9	14
5	10	15



row-major
order

column-major
order

L2-2.7-Transpose Part 2

Transpose

array
matrix
image
data structures

```
struct foo {  
    float f;  
    int i;  
};  
foo array[1000];
```

array of
structures { f f f f f f f f f f f f f f f f } ... AoS

{}{}{} transpose

structure
of arrays { f f f f f f f f } ... { i i i i i i i i } ... SoA

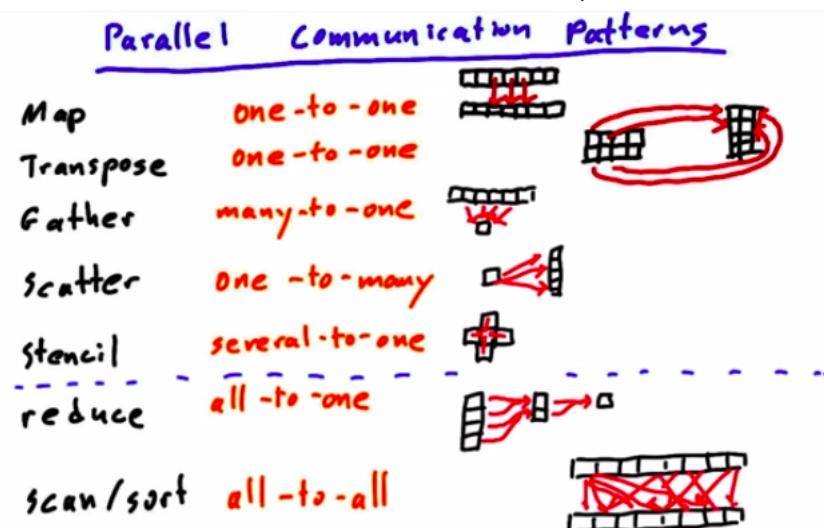
L2-2.8-What Kind of Communication Pattern - Quiz

Quiz: Label code snippets by pattern

```
float out[], in[];  
int i = threadIdx.x;  
int j = threadIdx.y;  
  
const float pi = 3.1415;  
  
out[i] = pi * in[i];  
out[i + j*128] = in[j + i*128];  
  
if (i % 2) {  
    out[i-1] += pi * in[i]; out[i+1] += pi * in[i];  
}  
    out[i] = (in[i] + in[i-1] + in[i+1]) * pi / 3.0f;
```

A. Map
B. Gather
C. Scatter
D. Stencil
E. Transpose

L2-2.9 Parallel Communication Patterns Recap



L2-2.10-Let Us Talk About GPU Hardware

✓ Parallel Communication Patterns

- How can threads efficiently access memory in concert?
 - How to exploit data reuse?
- How can threads communicate partial results by sharing memory?
 - Safely?

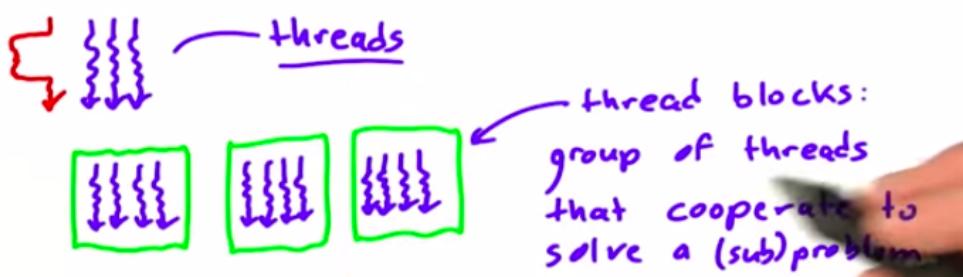
⇒ GPU Hardware.

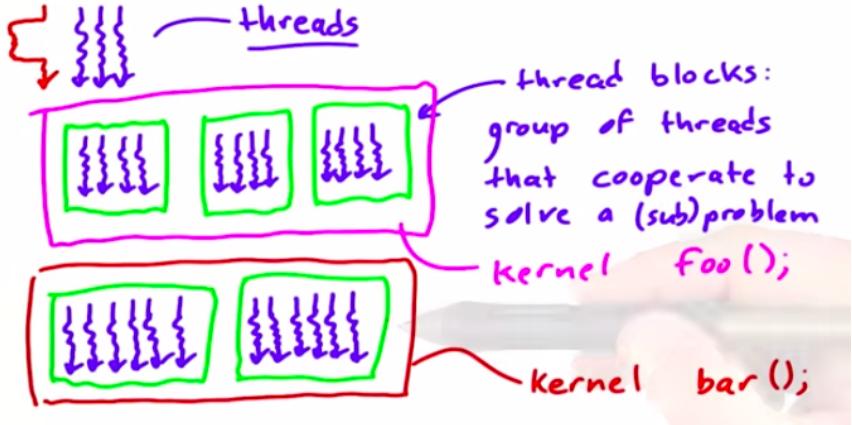


L2-2.11-Programmer View of the GPU

Summary of programming model

kernels - C / C++ function

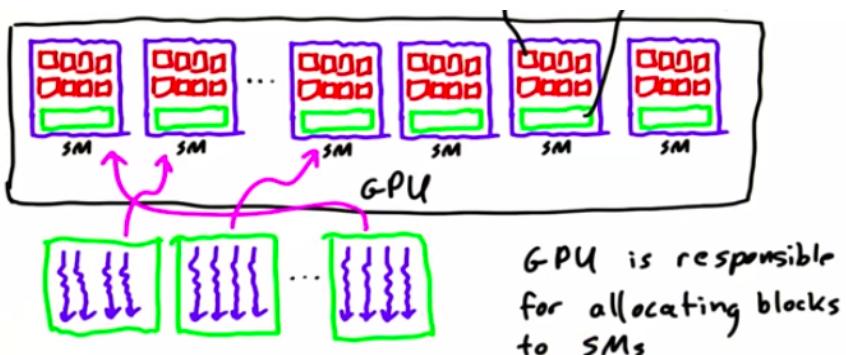
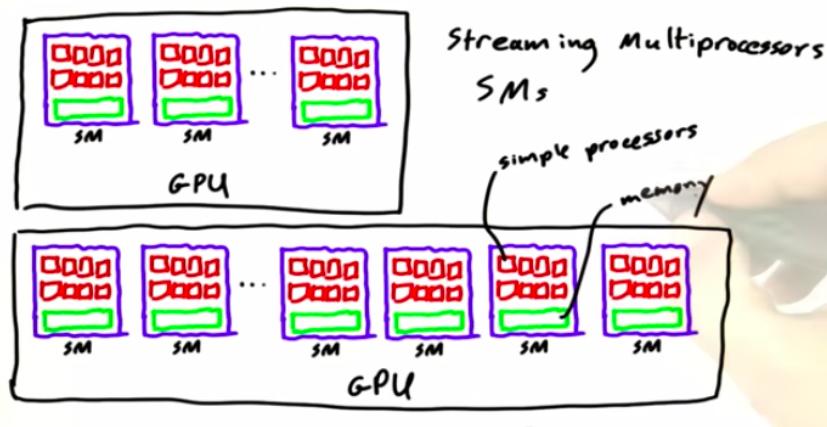




L2-2.12-Thread Blocks And GPU Hardware

Thread Blocks and GPU Hardware

- Why do we divide the problem into blocks?
- When are blocks run?
- In what order do they run?
- How can threads cooperate?
- with what limitations?



L2-2.13-Thread and Blocks - Quiz

Quiz

- A thread block contains many threads
- An SM may run more than one block
- A block may run on more than one SM
- All the threads in a thread block may cooperate to solve a subproblem
- All the threads that run on a given SM may cooperate to solve a subproblem

L2-2.14-Another Quiz on Thread Blocks

Quiz

The ~~O programmer~~ is responsible for defining
~~O GPU~~ thread blocks in software.

The ~~O programmer~~ is responsible for allocating
~~O GPU~~ thread blocks to hardware streaming
multiprocessors (SMs)

L2-2.15-What Can The Programmer Specify-Quiz

Quiz

Given a single kernel that is launched on many thread blocks including X and Y, the programmer can specify ...

- that Block X will run at the same time as Block Y
- that Block X will run after Block Y
- that Block X will run on SM Z

L2-2.16-CUDA Makes Few Guarantees About Thread Block

CUDA makes few guarantees about when and where thread blocks will run.

Advantages

- hardware can run things efficiently
- no waiting on slowpokes
- scalability!
 - from cell phones to supercomputers
 - from current to future GPUs

CUDA makes few guarantees about when and where thread blocks will run.

Advantages

flexibility → efficiency

Consequences

no assumptions blocks → SM

scalability

no communication between blocks
"dead lock"

threads, blocks must complete

L2-2.17-A Thread Block Programming Example (-Quiz?)

```

1 #include <stdio.h>
2
3 #define NUM_BLOCKS 16
4 #define BLOCK_WIDTH 1
5
6 __global__ void hello()
7 {
8     printf("Hello world! I'm a thread in block %d\n", blockIdx.x);
9 }
10
11
12 int main(int argc,char **argv)
13 {
14     // launch the kernel
15     hello<<<NUM_BLOCKS, BLOCK_WIDTH>>>();
16
17     // force the printf()s to flush
18     cudaDeviceSynchronize();
19
20     printf("That's all!\n");
21
22     return 0;
23 }
```

Quiz: How many different outputs can different runs of this program produce?

- 1
- 26
- 65,536
- 21 trillion.

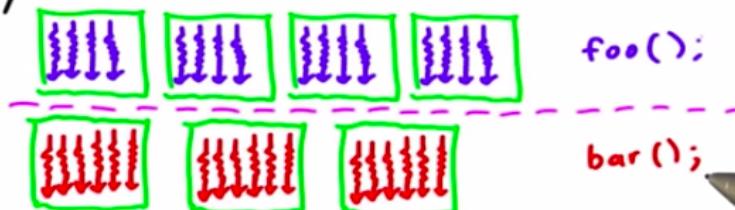
L2-2.18-A Programming Example on Blocks

```
1 #include <stdio.h>
2
3 #define NUM_BLOCKS 16
4 #define BLOCK_WIDTH 1
5
6 __global__ void hello()
7 {
8     printf("Hello world! I'm a thread in block %d\n", blockIdx.x);
9 }
10
11
12 int main(int argc,char **argv)
13 {
14     // launch the kernel
15     hello<<<NUM_BLOCKS, BLOCK_WIDTH>>>();
16
17     // force the printf()s to flush
18     cudaDeviceSynchronize();
19
20     printf("That's all!\n");
21
22     return 0;
23 }
```

L2-2.19-What Does CUDA Guarantee

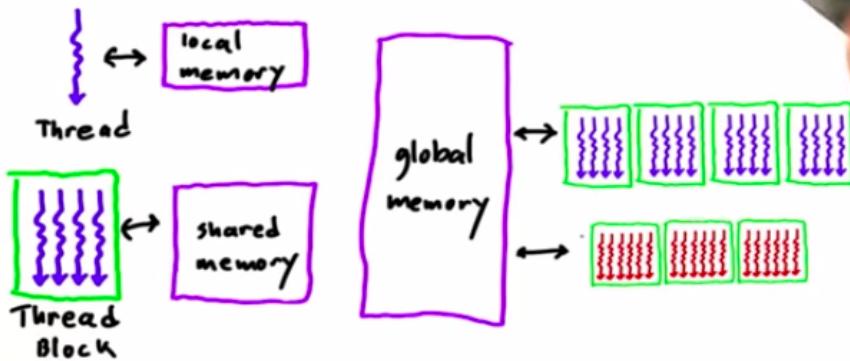
CUDA guarantees that:

- all threads in a block run on the same SM at the same time
- all blocks in a kernel finish before any blocks from the next kernel run



L2-2.20-GPU Memory Model

MEMORY MODEL



L2-2.21-A Quiz about GPU Memory Model

Quiz

- All threads from a block can access the same variable in that block's shared memory
- Threads from two different blocks can access the same variable in global memory
- Threads from different blocks have their own copy of local variables in local memory
- Threads from the same block have their own copy of local variables in local memory

L2-2.22-Synchronization-Barrier

Synchronization

threads can access each other's results through shared and global memory

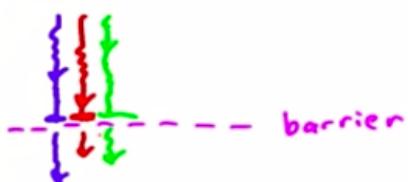
⇒ they can work together!

Danger: what if a thread reads a result before another thread writes it?

Threads need to synchronize

Barrier - point in the program where threads stop and wait.

when all threads have reached the barrier, they can proceed.



L2-2.23-The Need For Barriers-Quiz

The need for barriers

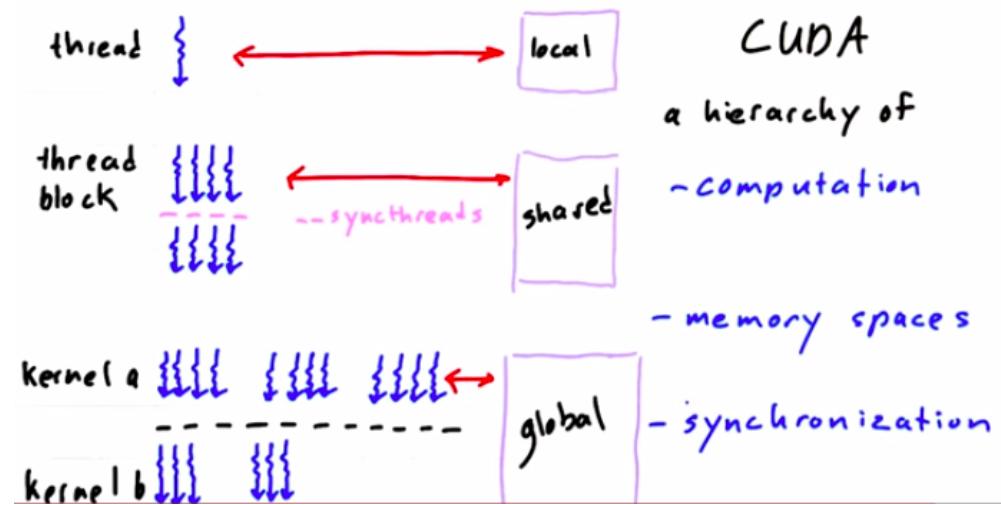
```

int idx = threadIdx.x
--shared-- int array[128];
array[idx] = threadIdx.x; Quiz: how many
if (idx < 127)           barriers does this
array[idx] = array[idx+1]; code need?
:

```



L2-2.24-Programming Model



L2-2.25-A Quiz on Synchronization - Quiz

Quiz : are following code snippets correct?

```

--global-- void foo (...) {
    --shared-- int s[1024];
    int i = threadIdx.x;
    --syncthreads();
    s[i] = s[i-1];
    --syncthreads();
    if (i%2) s[i] = s[i-1];
    --syncthreads();
     { s[i] = (s[i-1] + s[i] + s[i+1]) / 3.0;
        printf ("s[%d]=%f\n", i, s[i]);
    }
}

```

L2-2.26-Writing Efficient Programs

Writing Efficient Programs

High-level strategies

1. Maximize arithmetic intensity

math
memory



- maximize compute ops per thread

- minimize time spent on memory per thread



L2-2.27-Minimize Time Spent On Memory

Minimize time spent on memory

Move frequently-accessed data to fast memory

local > shared >> global >> CPU
"host"

L2-2.28-Global Memory

L2-2.29-Shared Memory : Note Instructor Notes!

Code Available:

<https://github.com/udacity/cs344/blob/master/Unit2%20Code%20Snippets/memory.cu>

L2-2.30-Quiz on Memory Access

Quiz

```

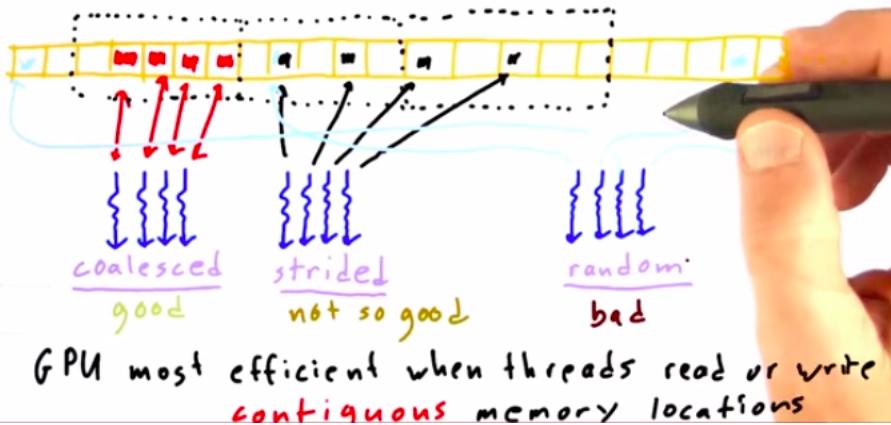
-- global -- void foo(float *x, float *y, float *z)
{
    -- shared -- float a,b,c;
    float s,t,u;
    :
    s = *x;      — Number the operations
    t = s;      — from fastest (1) to
    a = b;      — slowest (4)
    *y = *z;      —
    :
}

```

Number the operations
from fastest (1) to
slowest (4)

L2-2.31-Coalesce Memory Access

Using coalesced global mem access



GPU most efficient when threads read or write contiguous memory locations

L2-2.32-A Quiz on Coalescing Memory Access

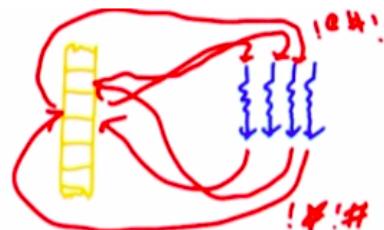
Quiz: Which statements have coalesced access pattern?

```
__global__ void foo(float *g)
{
    float a = 3.14;
    int i = threadIdx.x;

 g[i] = a;
 g[i*2] = a;
 a = g[i];
 a = g[BLOCK_WIDTH/2 + i];
 g[i] = a * g[BLOCK_WIDTH/2 + i];
 g[BLOCK_WIDTH-1 - i] = a;
}
```

L2-2.33-A related Problem Part 1

Problem: lots of threads reading and writing same memory locations



10,000 threads incrementing 10 array elements

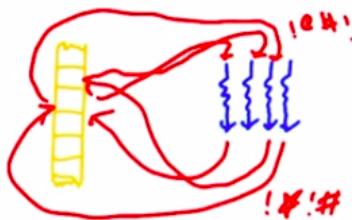
L2-2.34-A related Problem Part 2

code Available:

<https://github.com/udacity/cs344/blob/master/Unit2%20Code%20Snippets/atomics.cu>

L2-2.35-Atomic Memory Operations

Problem: lots of threads reading and writing same memory locations



10,000 threads incrementing 10 array elements

⇒ wrong answer!

Solution: atomic memory operations!

Atomsics: atomic Add() atomic Min()
XOR()
atomic CAS() - compare-and-swap

code Available:

<https://github.com/udacity/cs344/blob/master/Unit2%20Code%20Snippets/atomics.cu>

L2-2.36-Limitations of Atomic Memory Operations

Limitations of atomics

- Only certain operations, data types - mostly int
⇒ work around using atomic CAS()

- Still no ordering constraints
⇒ floating-point arithmetic non-associative!
 $(a+b)+c \neq a+(b+c)$ for float
 $a = 1, b = 10^{99}, c = -10^{99}$

- serializes access to memory
⇒ SLOW!

L2-2.37-Timing Atomic Operations

Code Available:

<https://github.com/udacity/cs344/blob/master/Unit2%20Code%20Snippets/atomics.cu>

L2-2.38-Let Us Time Some Code-Quiz

same Code Available:

<https://github.com/udacity/cs344/blob/master/Unit2%20Code%20Snippets/atomics.cu>

Quiz / Programming Exercise

Modify the code to time:

- 10^6 threads incrementing 10^6 elements
- 10^6 threads atomically incrementing 10^6 elements
- 10^6 threads incrementing 100 elements
- 10^6 threads atomically incrementing 100 elements
- 10^7 threads atomically incrementing 100 elements

correct? rank (1=fastest, 5=slowest)

L2-2.39-High Arithmetic Intensity

Strategies for efficient CUDA programming

1. High arithmetic intensity $\frac{\text{math}}{\text{memory}}$
 - minimize time spent on memory
 - put data in faster memory
 $\text{local} > \text{shared} > \text{global}$
 - use coalesced global memory access
adjacent threads access contiguous chunk of memory

2. Avoid thread divergence

```
if ( condition )
{
    some code
}
else
{
    some other code
}
:
```

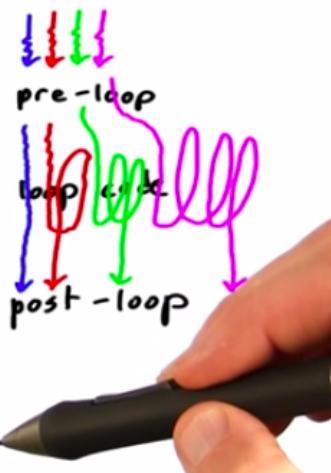


L2-2.40-Thread Divergence

```

for (int i=0; i<=threadIdx; ++i)
{
    ... some loop code ...
}
:
post-loop code
pre-loop  loop  loop  loop  loop  post-loop
-----+-----+-----+-----+-----+
        |       |       |       |       |
        +-----+-----+-----+-----+

```



L2-2.41 Summary of Unit 2

Summary

- Communication patterns
 - gather, scatter, stencil, transpose
 - GPU hardware & programming model
 - SIMDs, threads, blocks, ordering
 - Synchronization
 - Memory model - local, global, shared, atomics
 - Efficient GPU programming
 - Access memory faster
 - coalescing global mem
 - use faster memory
 - Avoid thread divergence

L2-2.43-Problem Set #2

Problem Set #2



Problem Set # 2

9	1	3	4	8
10	4	6	0	10
2	4	1	2	7
2	3	6	1	
8	0	0	3	9

Avg Intensity Value =

$$\frac{1}{9} (4 + 10 + 4 + 6 + 2 + 1 + 2 + 3)$$

$$(w_1 \cdot 4) + (w_2 \cdot 10) + (w_3 \cdot 4) + (w_4 \cdot 6).$$

Blur Examples

Weighted

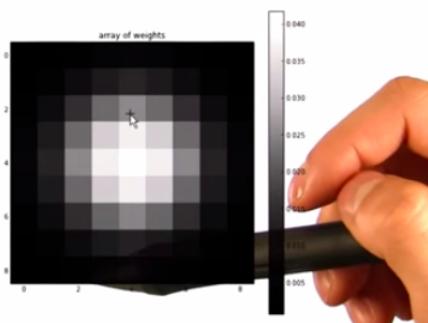


Unweighted.



Weights

.1	.1	.1	.1
.1	.3	.3	.1
.1	.3	.3	.1
.1	.1	.1	.1



What You Need To Do

- ① Write the blur kernel
- ② Write Kernel to Separate Color Image to R, G, B channels
- ③ Allocate memory for the filter
- ④ Set Grid and Block Size.