

CS344 Introduction to Parallel Programming

Lesson 5: Optimizing GPU Programs

L5-5.1-Optimizing GPU Programs

Optimizing GPU Programs

Goal: solve problems faster

- solve bigger problems
- solve more problems

Good news: first port often gets a speedup

GPU programmer \equiv care about perf!

Quiz: principals of efficient GPU programming

- decrease arithmetic intensity
- decrease time spent on memory operations
- coalesce global memory accesses
- do fewer memory operations per thread
- avoid thread divergence
- move all data to shared memory

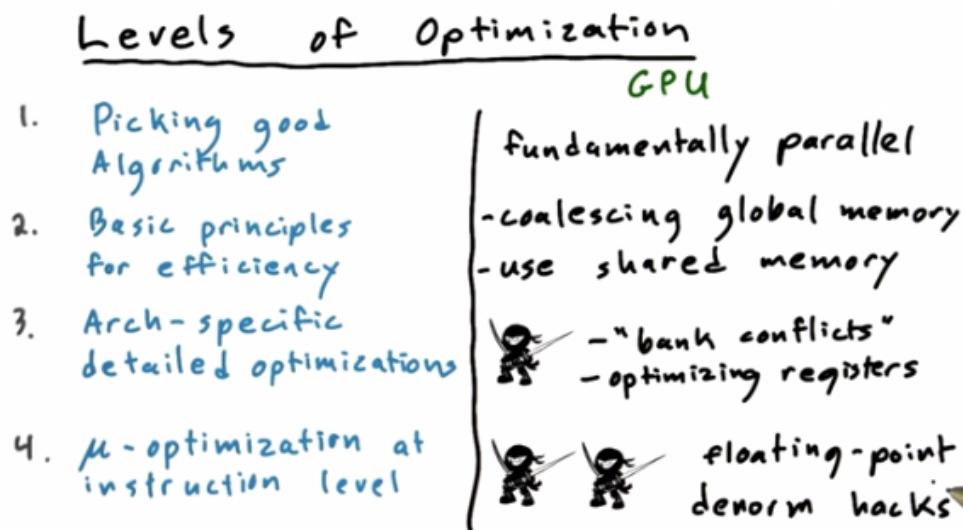
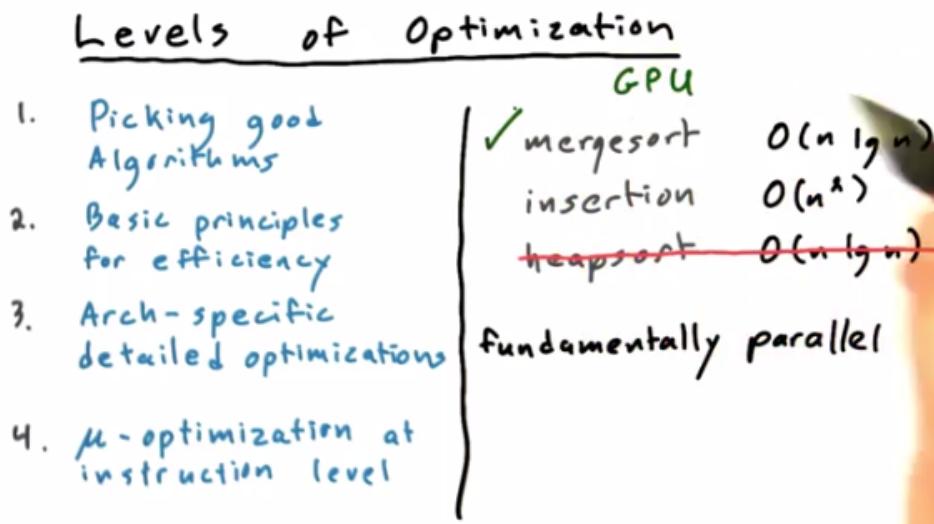
L5-5.2-Levels of Optimization Part 1

Levels of Optimization

Quiz

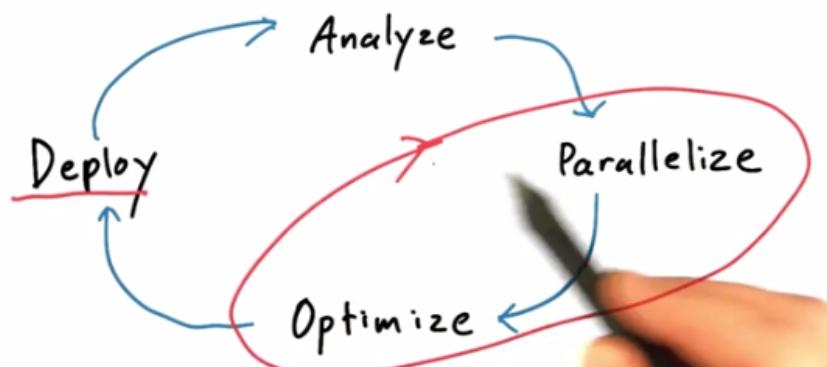
1. Picking good algorithms
 - Vector registers - SSE, AVX
2. Basic principles for efficiency
 - Use mergesort $O(n \lg n)$ vs insertion sort $O(n^2)$
3. Arch-specific detailed optimizations
 - Write cache-aware code
 - e.g. traverse rows & cols
 - Block for the L1 cache
 - `float recipSqrt = (float) 0x 5F3759D8 - (a >> 1);`
4. μ -optimization at instruction level

L5-5.3-Levels of Optimization Part 2



L5-5.4-APOD Part 1

APOD - Systematic Optimization



L5-5.5-APOD Part 2

APOD - Systematic Optimization

Analyze: Profile whole application

- where can it benefit?

- by how much?

Parallelize: Pick an approach



APOD - Systematic Optimization

Analyze: Profile whole application

- where can it benefit?

- by how much?

Parallelize: Pick an approach
 ↳ Pick an algorithm ← important!

Optimize: Profile-driven optimization ← measure!

Deploy: Don't optimize in a vacuum!
 "make it real" "small" speedups help

L5-5.6-Weak vs Strong Scaling

Weak vs. Strong Scaling

Serial program solves problem size P in time T

ex: fold a protein in an hour

Weak scaling: run a larger problem (or more)

ex: fold a bigger protein, or more small ones

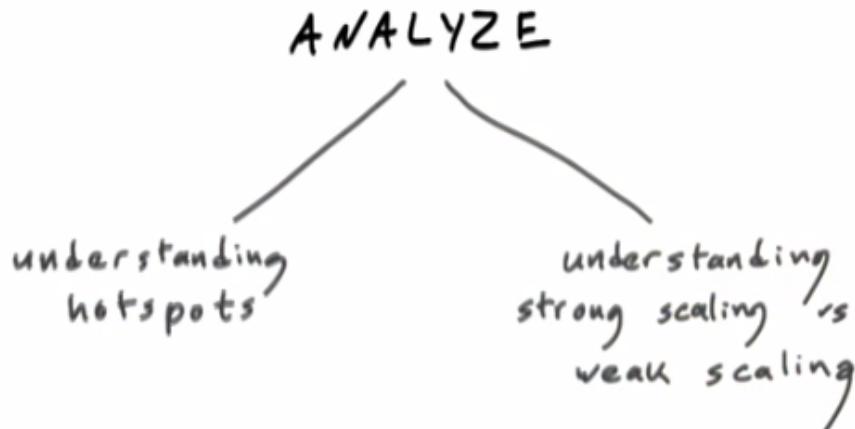
solution size varies with fixed problem size per core

Strong scaling: run a problem faster

ex: fold the same protein in a minute

solution size varies with fixed total problem size

L5-5.7-Analyze



Understanding Hotspots

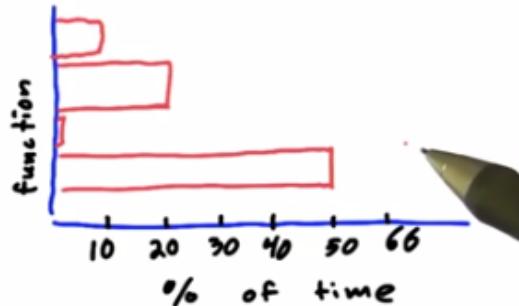
Don't rely on intuition!

Run a profiler

gProf

VTune

VerySleepy



Quiz: max speedup possible parallelizing top fn? X

top 3 fns? X

L5-5.8-Understanding Hotspots

Understanding Hotspots

"Amdahl's Law"

Total speedup from parallelization is limited
by portion of time spent doing
something to be parallelized

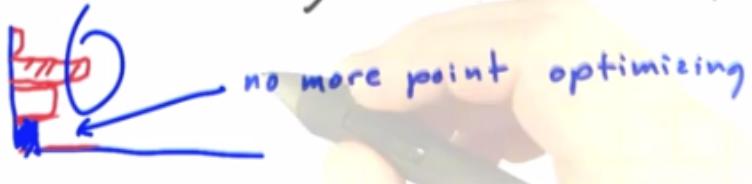
$$\text{max speedup} \rightarrow \frac{1}{1-p} \quad p \equiv \% \text{ parallelizable time}$$

Understanding Hotspots

Know these limits!

ex: 50% \Rightarrow 2x speed up max
 \Rightarrow Refactor!

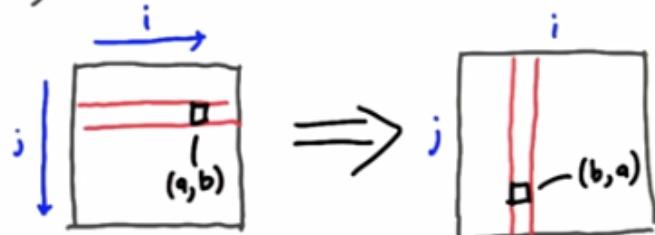
ex: read/write larger batches of data



L5-5.9-Parallelize

PARALLELIZE

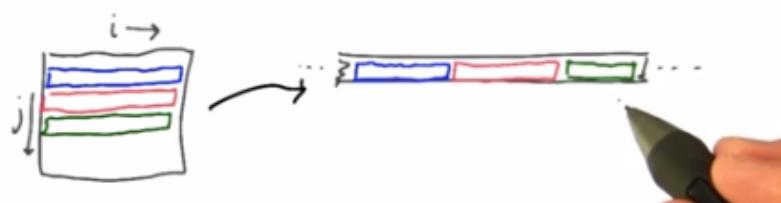
Running example: Matrix transpose



Restrict to square matrices

L5-5.10-Transpose Code Example Part 1

Transpose - code example



L5-5.11-Transpose Code Example Part 2

L5-5.12-Transpose Code Example Part 3

L5-5.13-Transpose Code Example Part 4

Transpose - code example

v1: serial

466 ms

v2: parallel per row

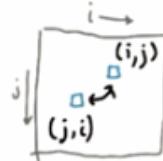
4.7 ms

v3: parallel per element

→ Programming exercise

- up to 1024 threads in block!

- use $K \times K$ threadblocks, $K=16$



Note: (I should not give) code available:

<https://github.com/udacity/cs344/blob/master/Unit5%20Code%20Snippets/transpose.cu>

L5-5.14-Maximum Parallelism is Not Always Best for Performance

Transpose - code example

v1: serial

466 ms

v2: parallel per row

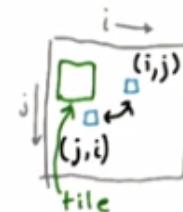
4.7 ms

v3: parallel per element

0.67 ms



maximum parallelism not
always best performance \Rightarrow "granularity
coarsening"



Transpose - code example

v1: serial

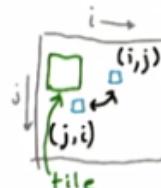
466 ms

v2: parallel per row

4.7 ms

v3: parallel per element

0.67 ms



Are we done? limits: memory

compute

device Query

L5-5.15-Theoretical Peak Bandwidth

From device query:

Memory clock: 2508 MHz

Memory bus: 128 bits

Quiz: what is the theoretical peak bandwidth?
_____ GB/s

L5-5.16-What Bandwidth Does Our Kernel Achieve

From device query:

Memory clock: $2508 \times 10^6 \frac{\text{clocks}}{\text{sec}}$

Memory bus: 128 bits = 16 bytes/clock

Quiz: what is the theoretical peak bandwidth?
40.128 GB/s

Quiz: what bandwidth does our kernel achieve?

$N=1024$ time = 0.67 ms _____ GB/s

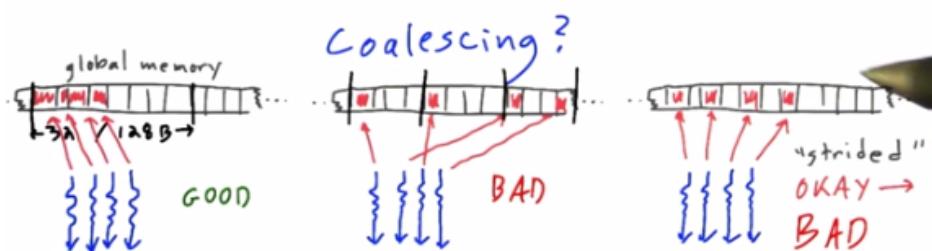
L5-5.17-Measuring Memory Usage of Our Transpose Code

Transpose - code example

v1: serial	466 ms	0.018 GB/s
v2: parallel per row	4.7 ms	1.8 GB/s
v3: parallel per element	0.67 ms	12.5 GB/s

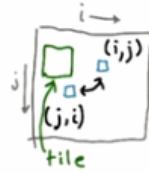
Transpose - code example DRAM utilization

v1: serial	466 ms	< 0.1%
v2: parallel per row	4.7 ms	4.5%
v3: parallel per element	0.67 ms	<u>31.1 %</u>



L5-5.18-Most GPU Programs Are Memory Limited

Transpose - code example		
v1: serial	466 ms	
v2: parallel per row	4.7 ms	
v3: parallel per element	0.67 ms	



most GPU codes are memory limited

⇒ always start by measuring bandwidth!

L5-5.19-Tools to Measure Bandwidth Utilization

Transpose - code example		DRAM utilization
v1: serial	466 ms	< 0.1%
v2: parallel per row	4.7 ms	4.5%
v3: parallel per element	0.67 ms	31.1%
→ bad coalescing on write		
Tools:	nSight	
	Linux/Mac: nSight Eclipse	
	Windows: nSight Visual Studio	
nvvp		

L5-5.20-Using NVVP Part 1

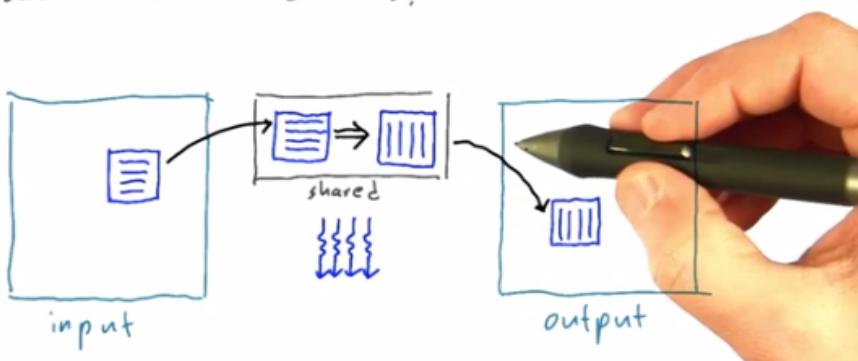
L5-5.21-Using NVVP Part 2

L5-5.22-Tiling

TILING

problem - coalesced reads, scattered writes

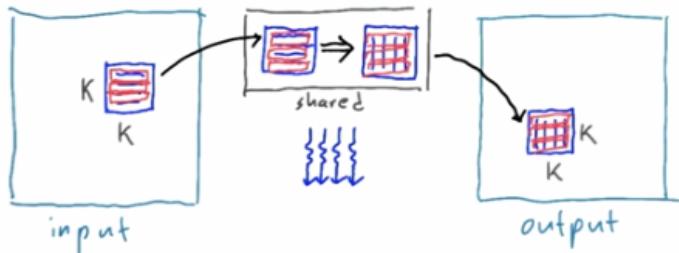
goal - coalesced reads, coalesced writes



TILING

Programming exercise

- declare array in shared memory
- copy appropriate elements in and out
 - ... in transposed fashion
- don't forget `--sync threads()`!



L5-5.23-Profiling the Tiling Code

NVVP analysis

L5-5.24-Littles Law

Goal: make code fast
→ utilize all the available memory bandwidth

"GPUs are fast!"

massively parallel extremely high-bandwidth memory

LITTLE'S LAW

-OPTIMIZE CUSTOMERS IN LINE AT STARBUCKS COFFEE



$$\text{NUMBER OF BYTES DELIVERED} = \text{AVERAGE LATENCY OF EACH TRANSACTION} \times$$

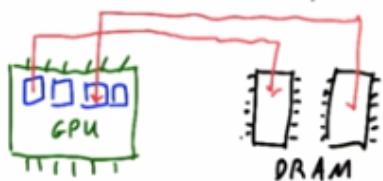
Quiz: what can we do to improve bandwidth?

- increase the number of bytes delivered
- increase the latency (time between transactions)
- decrease the number of bytes delivered
- decrease the latency (time between transactions)

L5-5.25-Littles Law for GPUs

Littles Law for GPUs

$$\text{useful bytes delivered} = \text{average Latency} * \text{bandwidth}$$



DRAM transactions: 100's cycles

- thread accessing global memory must wait
- have many threads in flight



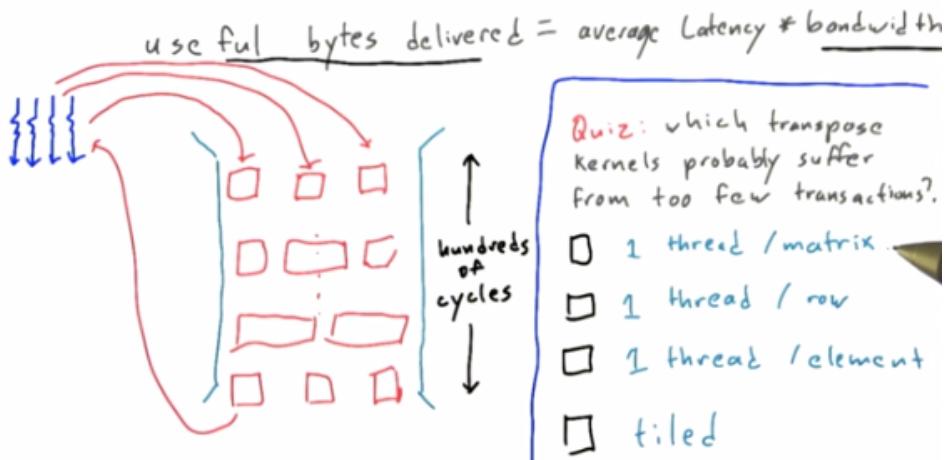
L5-5.26-Useful Bytes Delivered

Littles Law for GPUs

$$\text{useful bytes delivered} = \text{average Latency} * \underline{\text{bandwidth}}$$



Littles Law for GPUs



L5-5.27-Reducing Average Wait Time Per Thread

Quiz: What can we do to reduce the average wait time per thread?

- eliminate the syncthreads call?
- reduce the number of threads per block?
- increase the number of threads per block?
- increase the number of blocks per SM?

L5-5.28-Occupancy Part 1

Occupancy

Each SM has a limited number of:

- thread blocks 8
- threads 1536 / 2048
- registers for all threads 65536
- bytes of shared memory 16 K - 48 K

Ex: 48K shared; kernel takes 16K \Rightarrow 3 blocks/SM

Ex: 1536 max threads; kernel takes 1024 \Rightarrow 1 block/SM

L5-5.29-Occupancy Part 2

Occupancy

Each SM has a limited number of:

- thread blocks 8
 - threads 1536 / 2048
 - registers for all threads 65536
 - bytes of shared memory 16 K - 48 K
- deviceQuery ↪

Occupancy

Quiz: How many thread blocks per SM can we run? — How many threads / SM? —

Which resources prevent us from running more?

- Max threads / SM?
- Max registers / SM?
- Max shared memory / SM?
- Max thread blocks / SM?

L5-5.30-Occupancy on Fermi GPUs

Occupancy - "Fermi"

Quiz: How many thread blocks per SM can we run? — How many threads / SM? —

Which resources prevent us from running more?

- Max threads / SM?
- Max registers / SM?
- Max shared memory / SM?
- Max thread blocks / SM?

L5-5.31-Occupancy on Daves Laptop and Udacity Servers

Occupancy

platform	max threads running	max threads possible	Occupancy
Dave's laptop	2048	2048	100%
Fermi (Udacity)	1024	1536	66%

CUDA Occupancy Calculator

L5-5.32-How to Affect Occupancy

Occupancy

How to affect occupancy?



- control amount of shared memory, e.g. tile size
- change # threads, blocks << , >>
- compilation options to control register usage

Increasing occupancy usually helps, to a point
 + exposes more parallelism, transactions in flight
 - may force GPU to run less efficiently

Transpose - code example

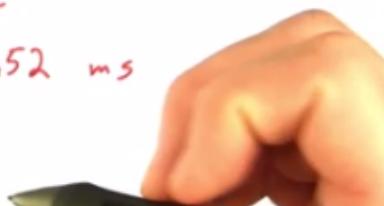
v1: serial	466 ms
v2: parallel per row	4.7 ms
v3: parallel per element	0.67 ms
v4: tiled	0.52 ms

Programming exercise: Try multiple tile sizes.
 Rank from fastest (1) to slowest (4): (on Udacity)

— 8x8 — 16x16 — 32x32 — 64x64

L5-5.33-Transpose Code Recap

Transpose - code example	DRAM utilization
v1: serial	466 ms
v2: parallel per row	4.7 ms
v3: parallel per element → bad coalescing on write	0.67 ms
v4: tiled 32x32 → threads waiting at barriers	0.52 ms
v5: tiled 16x16	.



L5-5.34-Transpose Code Memory Utilization

Transpose - code example	DRAM utilization
v1: serial	82.7 ms 0.07 %
v2: parallel per row	3.2 ms 1.76 %
v3: parallel per element → bad coalescing on write	0.35 ms 16.1 %
v4: tiled 32x32 → threads waiting at barriers	0.33 ms 17.1 %
v5: tiled 16x16 → shared memory bank conflicts	0.13 ms 43.5 %
v6: tiled, padded 16x17	0.126 ms 45.0 %

L5-5.35-Optimizing Compute Performance

Optimizing Compute Performance

Goal: maximize useful computation / second

- ✓ - minimize time waiting at barriers
- minimize thread divergence



L5-5.36-Minimize Thread Divergence

Minimizing thread divergence

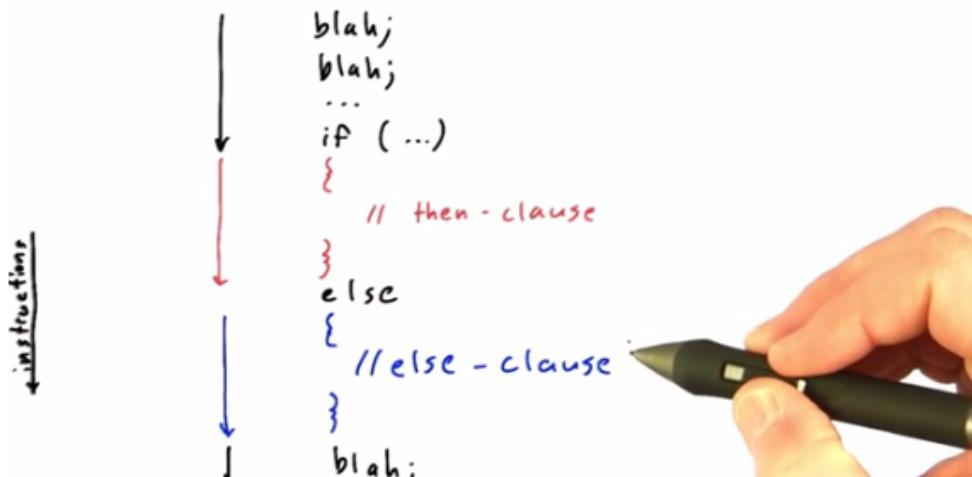
warp - set of threads that execute the same instruction at a time

SIMD - Single Instruction, Multiple Data

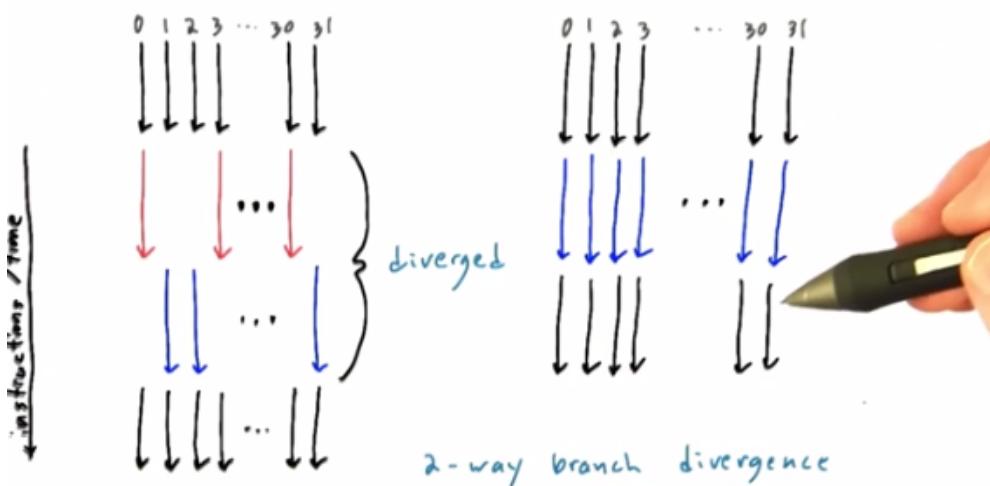
CPU: SSE / AVX vector registers

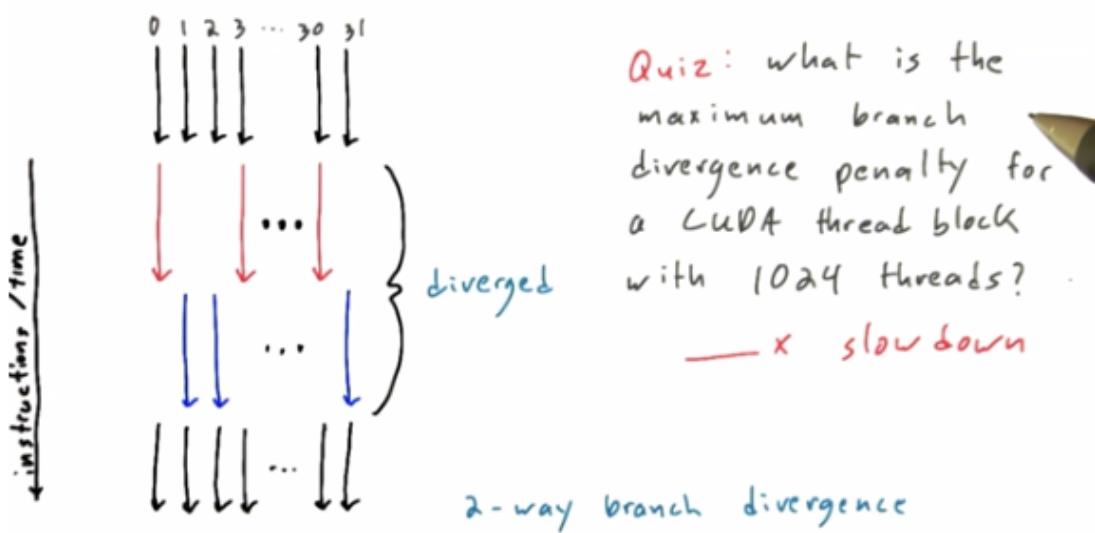
SIMT - Single Instruction, Multiple Thread

what happens at a branch?



L5-5.37-Thread Divergence Penalty





L5-5.38-Switch Statements and Thread Divergence Part 1

```
switch ( expr )
{
    case 1: ... break;
    case 2: ... break;
    case 3: ... break;
    :
    case 32: ... break;
}
```



Quiz: what will be the slowdown for each of the following expressions in switch statement?

- 32 `switch (threadIdx.x % 32) {case 0..31} ...`
`Kernel <<< 1024, 1 >>> ();`
- `switch (threadIdx.x % 64) {case 0..63} ...`
`Kernel <<< 1024, 1 >>> ();`
- `switch (threadIdx.y) {case 0..31} ...`
`Kernel <<< 64x16, 1 >>> ();`
- `switch (threadIdx.y) {case 0..31} ...`
`Kernel <<< 16x16, 1 >>> ();`

L5-5.39-Switch Statements and Thread Divergence Part 2

Quiz: what will be the slowdown for each of the following expressions in switch statement?

— switch (threadIdx.x % 2) { case 0..3 } ...
kernel <<<1024, 1 >>> ();

— switch (threadIdx.x / 32) { case 0..3 } ...
kernel <<<1024, 1 >>> ();

— switch (threadIdx.x / 8) { case 0..63 } ...
kernel <<<1024, 1 >>> ();

L5-5.40-Loops and Thread Divergence

Loops

```
--global-- void foo(...){  
    ...  
    for (int i=0; i < threadIdx.x%32; i++) { bar();}  
    for (int i=0; i < threadIdx.x/32; i++) { bar();}  
}  
foo <<< 1024, 1 >>> (...);
```

expensive!

Quiz: which is faster, loop 1 or loop 2? —
About how much faster? —

L5-5.41-Thread Divergence in the Real World Part 1

Branch divergence in the real world

Ex: operating on 1024 x 1024 image
special handling of pixels on the boundary

```
--global-- void perPixelKernel (float3 *image)  
{  
    if (threadIdx.x == 0 || threadIdx.x == 1024  
        || threadIdx.y == 0 || threadIdx.y == 1024) {  
        ... deal with boundary conditions ...  
    } else {  
        ... do something to pixel ...  
    }  
}
```

Branch divergence in the real world

Ex: operating on 1024 x 1024 image
special handling of pixels on the boundary

Quiz: what is the maximum branch divergence of any warp? —-way

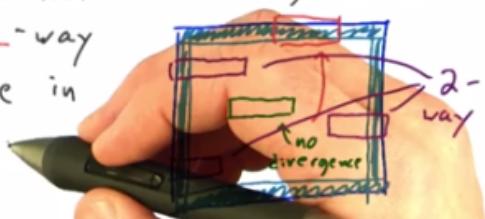
L5-5.42-Thread Divergence in the Real World Part 2

Branch divergence in the real world

Ex: operating on 1024 x 1024 image
special handling of pixels on the boundary

Quiz: what is the maximum branch divergence of any warp? 2-way

How many threads are in diverged warps? —



L5-5.43-Thread Divergence in the Real World Part 3

Branch divergence in the real world

- Be aware of branch divergence

- But don't freak out about it

Reducing branch divergence

- Avoid branchy code

- lots of if, switch stmts?

- adjacent threads likely to take different paths?

→ try to restructure

- Beware large imbalance in thread workloads

- look at loops, recursive calls

→ try to restructure

L5-5.44-Assorted Math Optimizations

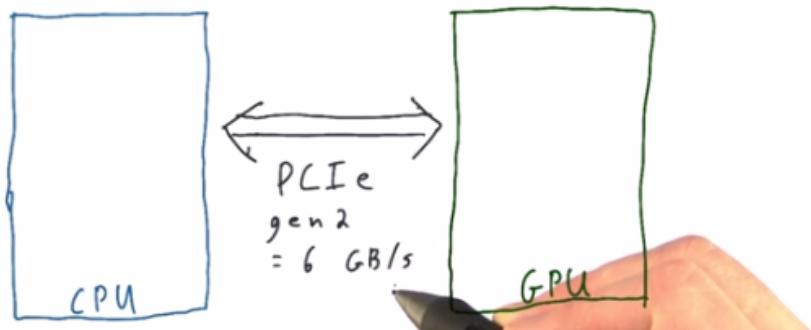
Assorted math optimizations

- use double precision only when you mean it
 $\text{fp64} > \text{fp32}$
 $\text{float } a = b + 2.5; > \text{float } a = b + 2.5f;$
- use intrinsics when possible
 $\text{__sin() } \text{__cos() } \text{__exp()}$ 2-3x less precision
than `math.h`, faster!
compiler flags for fast math

L5-5.45-Host-GPU Interaction

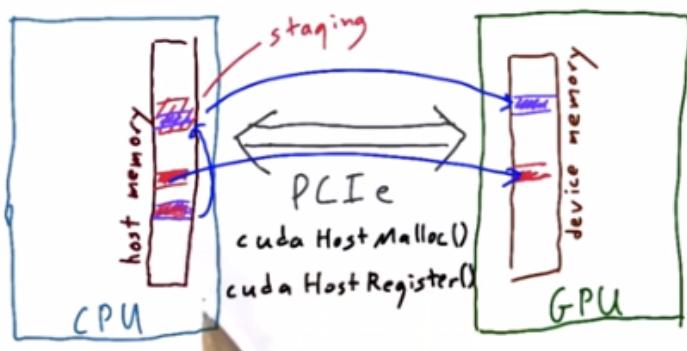
Host - GPU interaction

Pinned (page-locked) host memory



Host - GPU interaction

Pinned (page-locked) host memory

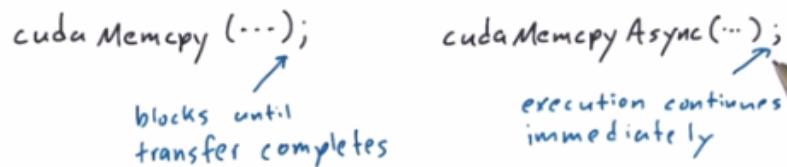


Host - GPU interaction

Pinned (page-locked) host memory

- faster!

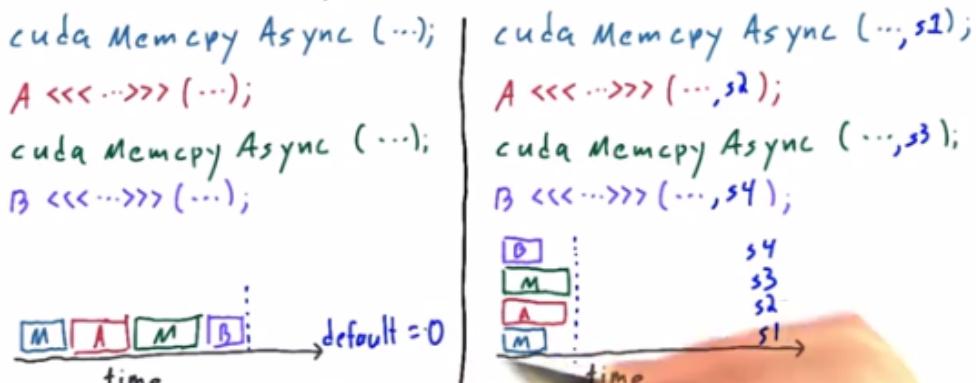
- enables `cudaMemcpyAsync()`



L5-5.46-Streams Part 1

Streams

stream: sequence of operations that execute in order
(memory transfers, kernels)



Streams

stream: sequence of operations that execute in order
(memory transfers, kernels)

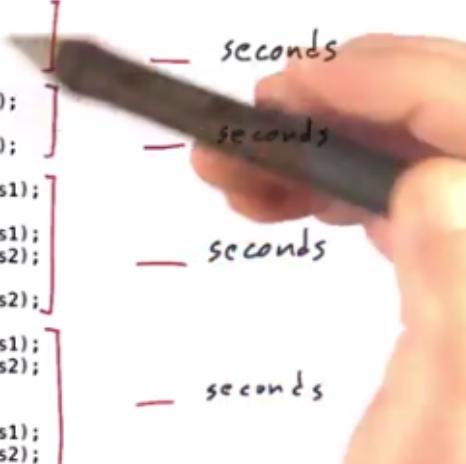
```
cudaStream_t s1;  
cudaStreamCreate(&s1);  
cudaStreamDestroy(s1);
```

asynchronous memory transfers → `cudaMemcpyAsync` → pinned memory

L5-5.47-Streams Part 2

Quiz: if all operations take 1 second, how long to complete:

```
cudaStream_t s1, s2;  
cudaStreamCreate(&s1); cudaStreamCreate(&s2);  
  
cudaMemcpy(&d_arr, &h_arr, numbytes, cudaMemcpyH2D);  
A<<<1, 128>>>(d_arr);  
cudaMemcpy(&h_arr, &d_arr, numbytes, cudaMemcpyD2H);  
  
cudaMemcpyAsync(&d_arr, &h_arr, numbytes, cudaMemcpyH2D, s1);  
A<<<1, 128, s1>>>(d_arr);  
cudaMemcpyAsync(&h_arr, &d_arr, numbytes, cudaMemcpyD2H, s1);  
  
cudaMemcpyAsync(&d_arr1, &h_arr1, numbytes, cudaMemcpyH2D, s1);  
A<<<1, 128, s1>>>(d_arr1);  
cudaMemcpyAsync(&h_arr1, &d_arr1, numbytes, cudaMemcpyD2H, s1);  
cudaMemcpyAsync(&d_arr2, &h_arr2, numbytes, cudaMemcpyH2D, s2);  
B<<<1, 192, s2>>>(d_arr2);  
cudaMemcpyAsync(&h_arr2, &d_arr2, numbytes, cudaMemcpyD2H, s2);  
  
cudaMemcpyAsync(&d_arr1, &h_arr1, numbytes, cudaMemcpyH2D, s1);  
cudaMemcpyAsync(&d_arr2, &h_arr2, numbytes, cudaMemcpyH2D, s2);  
A<<<1, 128, s1>>>(d_arr1);  
B<<<1, 192, s2>>>(d_arr2);  
cudaMemcpyAsync(&h_arr1, &d_arr1, numbytes, cudaMemcpyD2H, s1);  
cudaMemcpyAsync(&h_arr2, &d_arr2, numbytes, cudaMemcpyD2H, s2);
```



Red brackets and lines are drawn from the code to indicate parallel execution times. The first section of operations (up to the first cudaMemcpyAsync) is grouped by a bracket and labeled 'seconds'. The second section (from the first cudaMemcpyAsync to the first cudaMemcpyAsync in the next group) is also grouped by a bracket and labeled 'seconds'. This pattern repeats for the entire sequence of operations.

L5-5.48-Streams Part 3

Quiz: if all operations take 1 second, how long to complete:

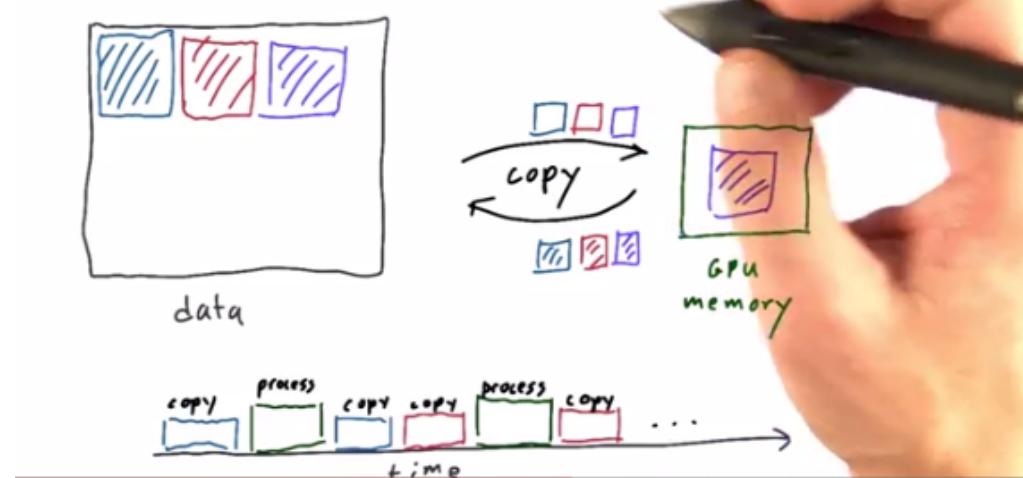
```
cudaStream_t s1, s2;  
cudaStreamCreate(&s1); cudaStreamCreate(&s2);  
  
cudaMemcpyAsync(&d_arr1, &h_arr1, numbytes, cudaMemcpyH2D, s1);  
A<<<1, 128, s2>>>(d_arr2);  
  
cudaMemcpyAsync(&d_arr1, &h_arr1, numbytes, cudaMemcpyH2D, s1);  
A<<<1, 128, s2>>>(d_arr1);
```



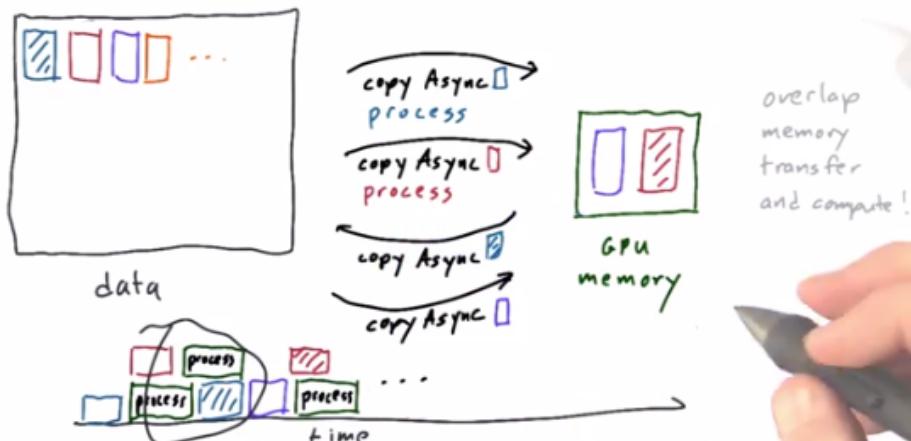
Red brackets and lines are drawn from the code to indicate parallel execution times. The first cudaMemcpyAsync operation is grouped by a bracket and labeled 'seconds'. The second cudaMemcpyAsync operation is also grouped by a bracket and labeled 'seconds'.

L5-5.49-Streams are Useful For

Streams



Streams



L5-5.50-Advantage of Streams

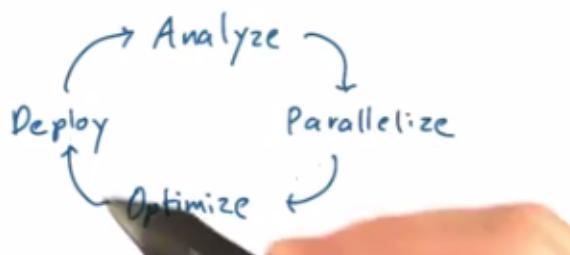
Streams

- Overlap memory & compute
- Help fill GPU with smaller kernels
 - many problems with limited parallelism
 - computations with narrow phases (reduce)
- Careats → see CUDA Programming Guide
 - streams
 - events

L5-5.51-Summary

Summary

- APOD
 - profile-guided
 - deploy early



Summary

- APOD

- measure & improve memory bandwidth
- minimize thread divergence
 - within warps
 - avoid branchy code
 - avoid thread workload imbalance
 - don't freak out

Summary

- APOD

- measure & improve memory bandwidth
- minimize thread divergence
- consider fast math
 - intrinsics `--sin()`, `--cos()`, etc
 - use double precision on purpose: $3.14 \neq 3.14f$

Summary

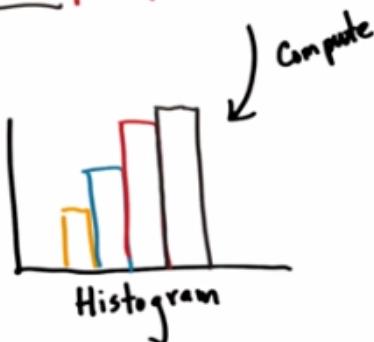
- APOD

- measure & improve memory bandwidth
- minimize thread divergence
- consider fast math
- use streams
 - overlap computation and CPU-GPU memory transfers

L5-5.52-Problem Set 5

problem Set #5

~~Fast~~



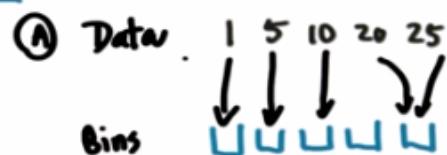
PS #3 → used
Atoms



Problem Set #5

① Experiment !!

② One Basic Strategy



Problem Set #5

Data: 412 318 302 018 724 655 749 780 201

bin ID: 41 31 30 01 72 65 74 73 20

coarse bin ID: 4 3 3 0 1 6 7 7 2

bin ID 41 31 30 01 72 65 74 73 20

Coarse bin ID	4	3	3	0	1	6	7	7	2
sorted BIN ID	-	-	-	-	-	-	-	-	-
Sorted coarse BIN ID	01	20	31	30	41	65	72	74	78

