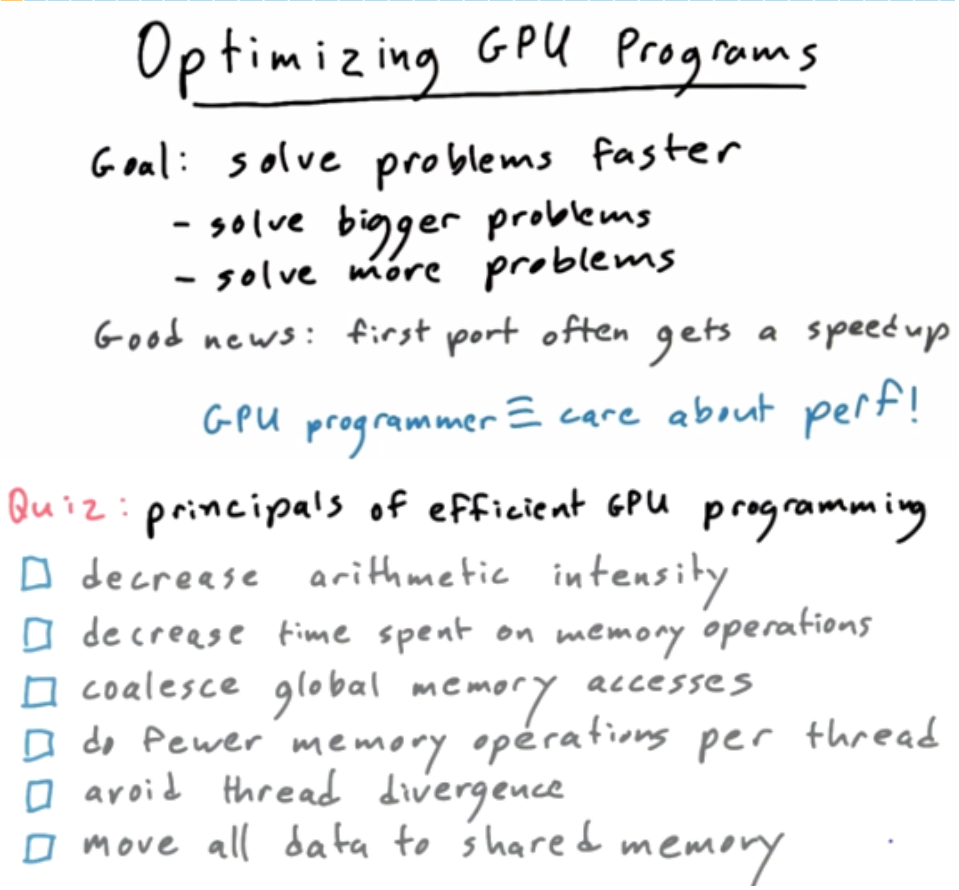
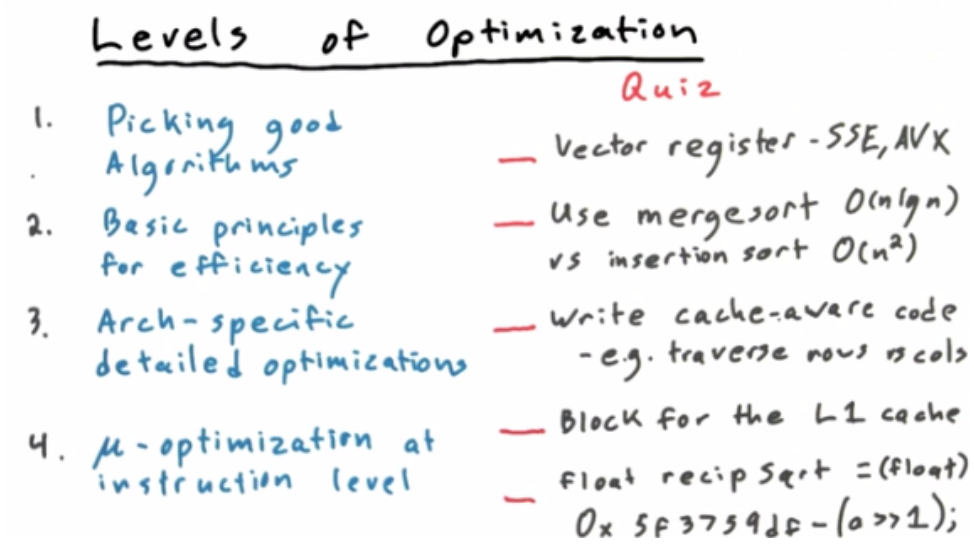
CS344 Introduction to Parallel Programming

Lesson 5:Optimizing GPU Programs

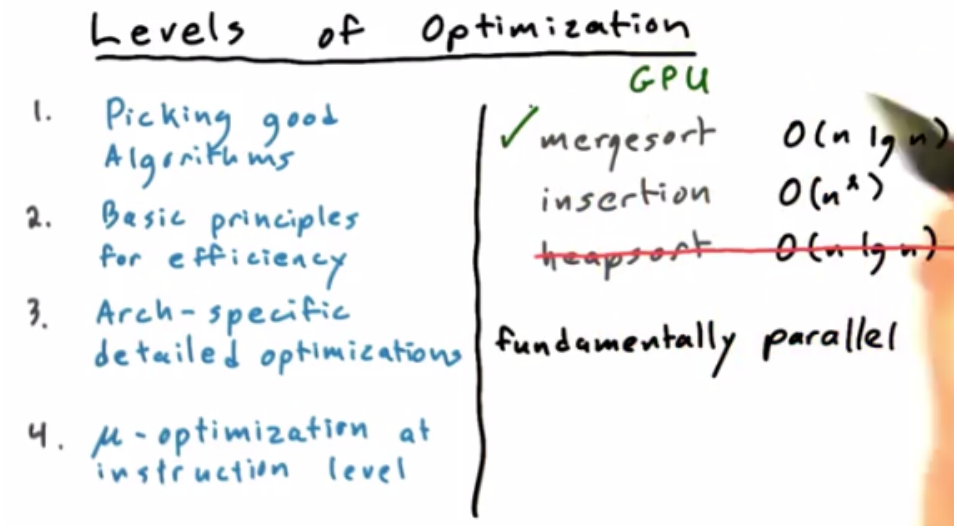
L5-5.1-Optimizing GPU Programs

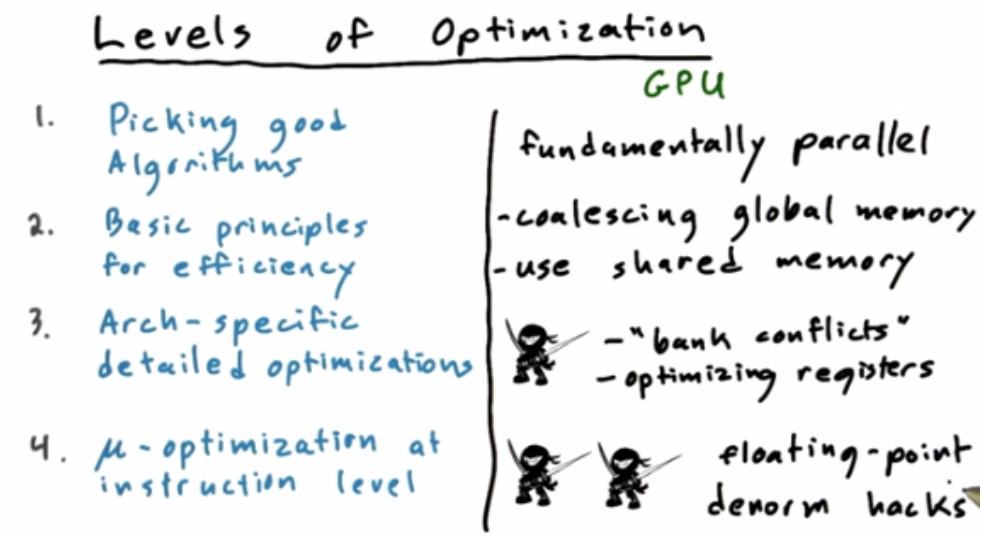


L5-5.2-Levels of Optimization Part 1

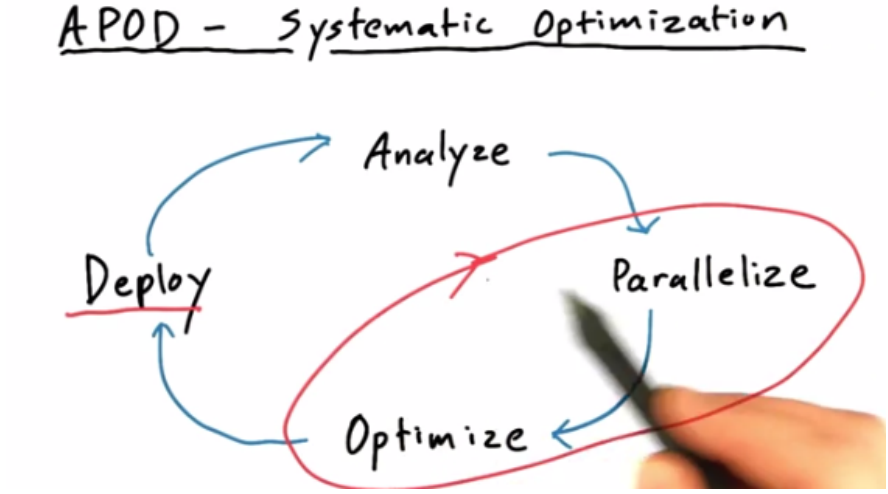


L5-5.3-Levels of Optimization Part 2



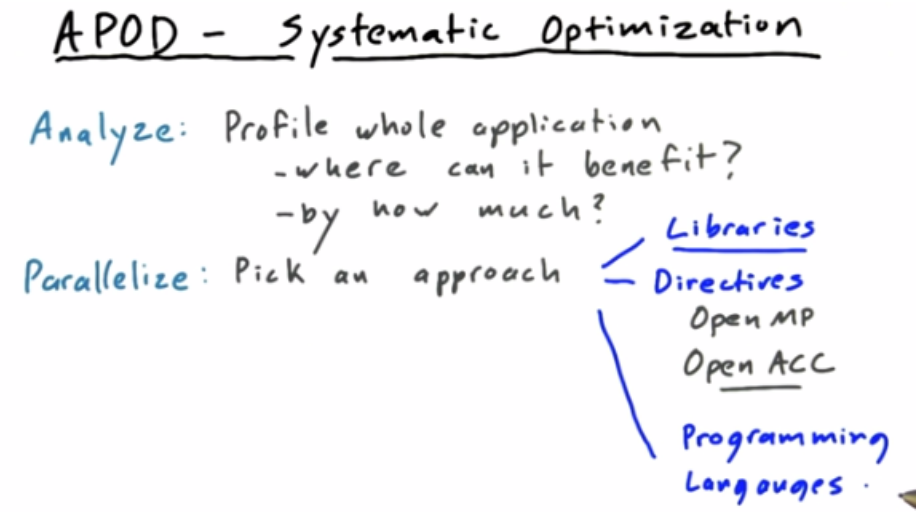


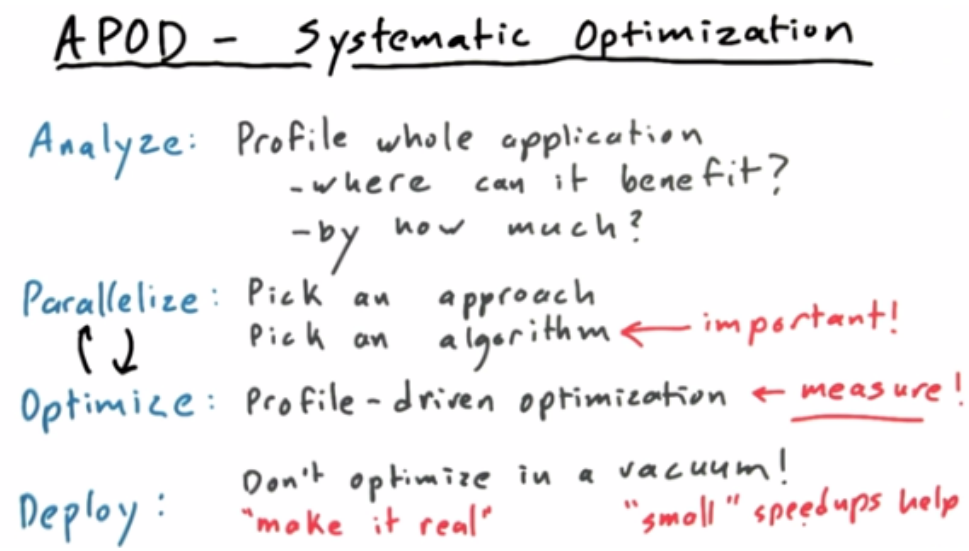
L5-5.4-APOD Part 1



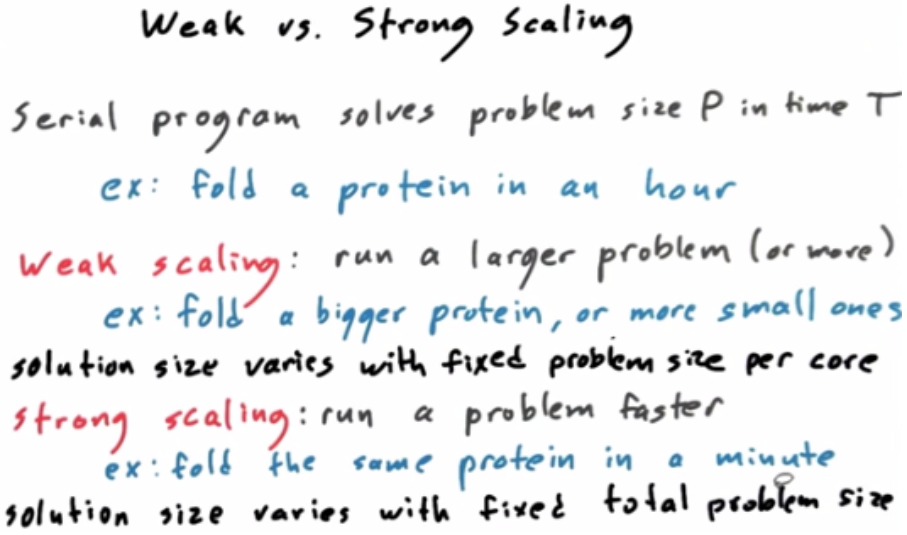
[CS344 Intro to Parallel Programming, Lesson 5]

L5-5.5-APOD Part 2

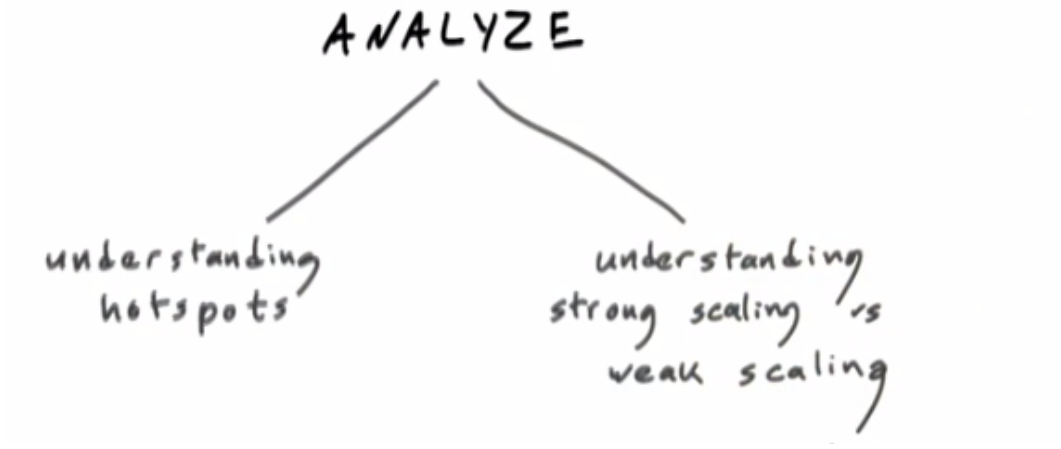


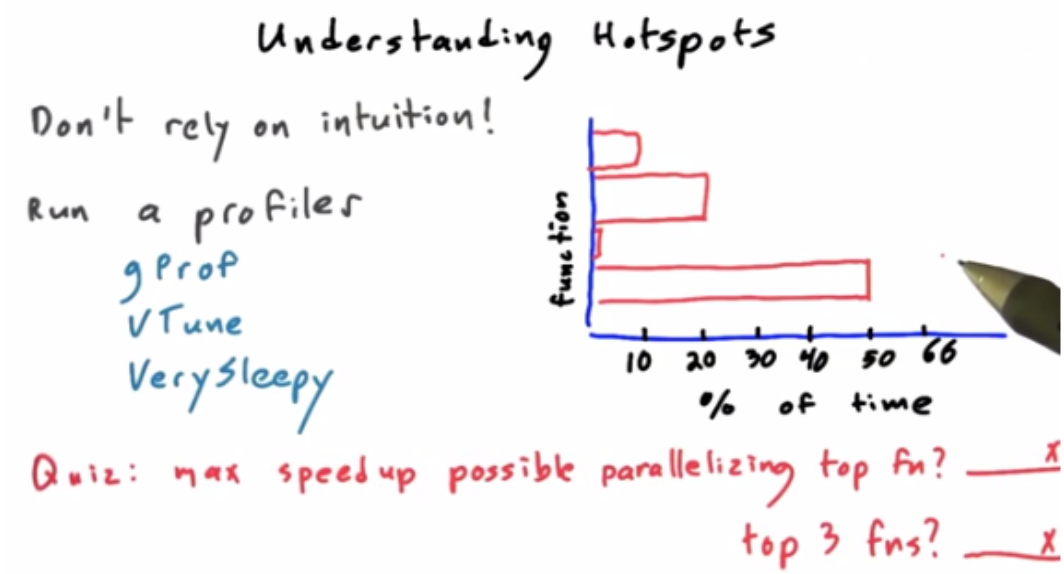


L5-5.6-Weak vs Strong Scaling

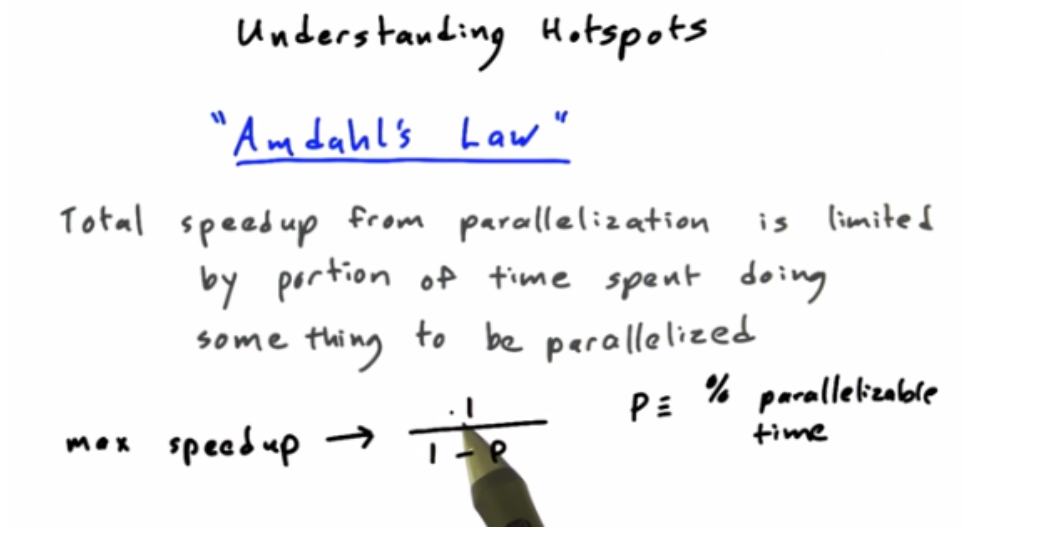


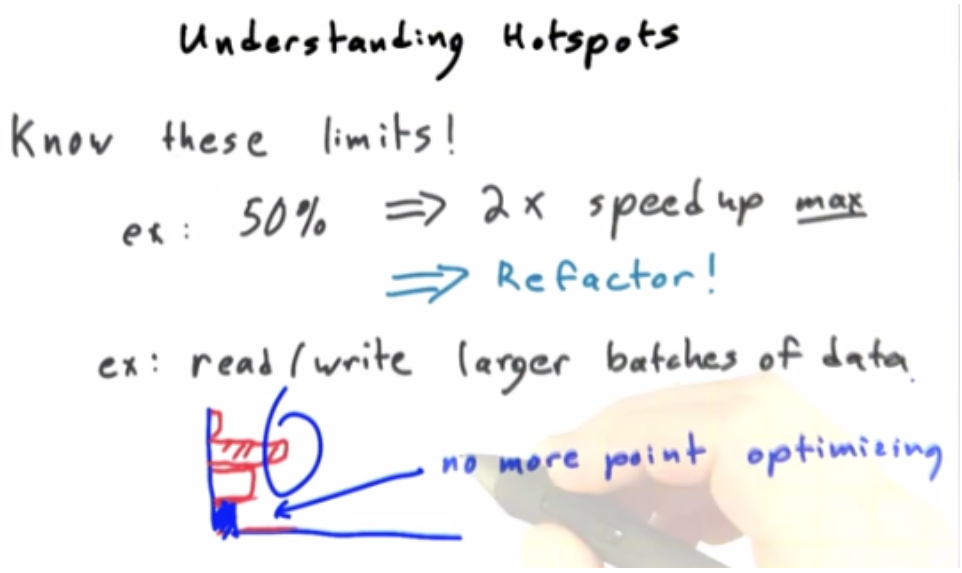
L5-5.7-Analyze



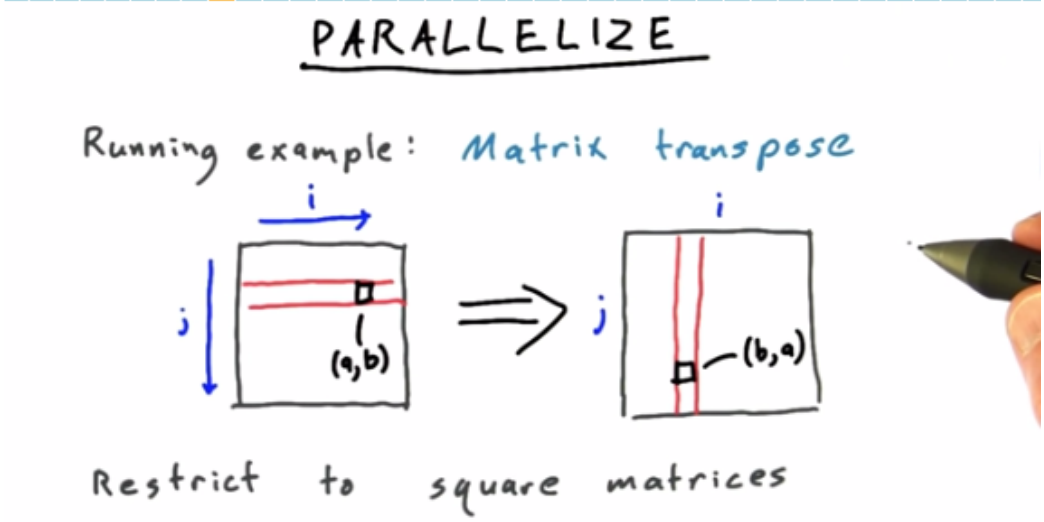


L5-5.8-Understanding Hotspots

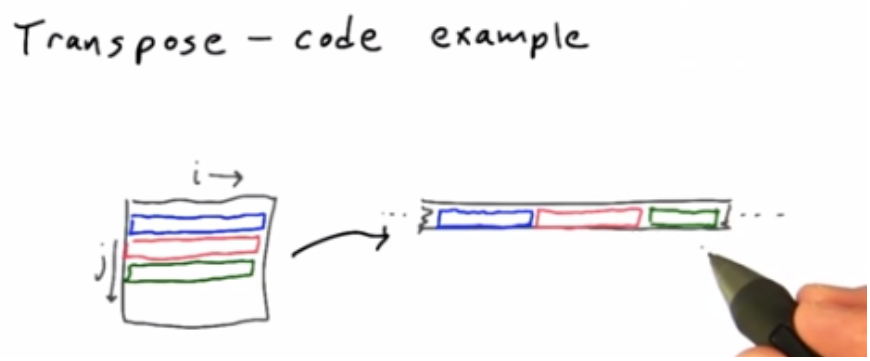




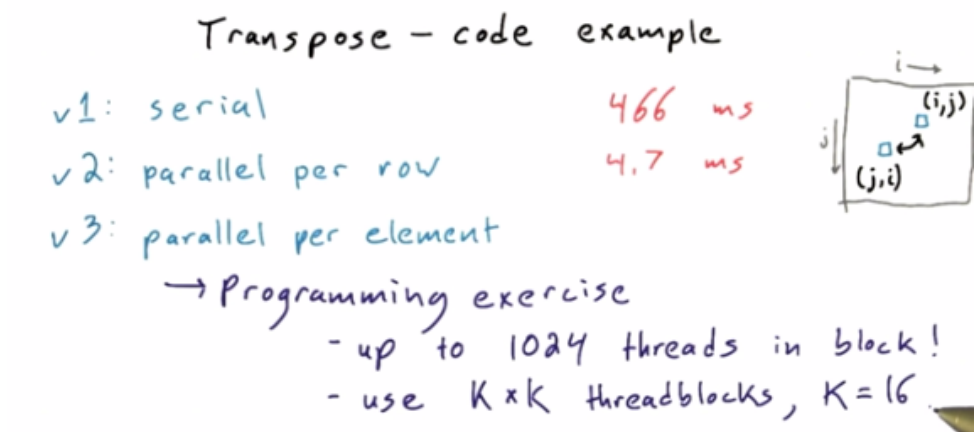
L5-5.9-Parallelize



L5-5.10-Transpose Code Example Part 1

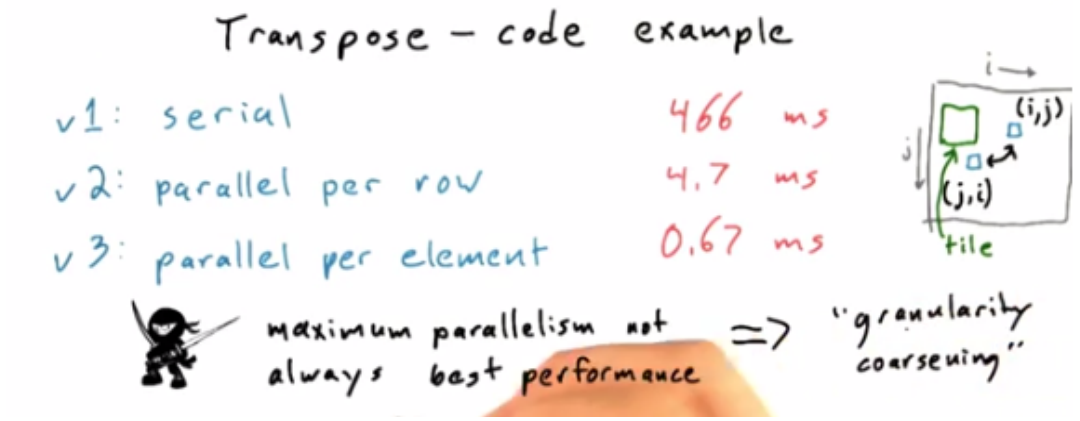


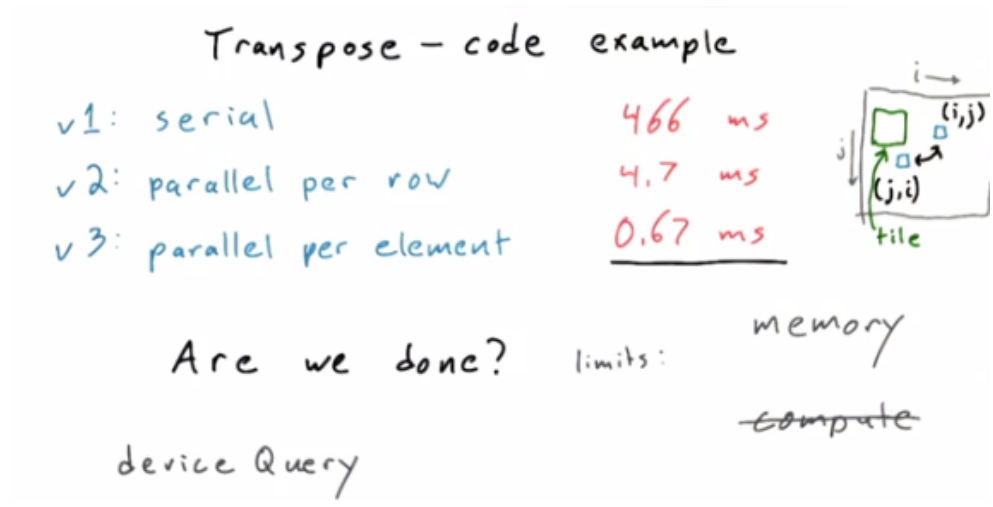
L5-5.11-Transpose Code Example Part 2
L5-5.12-Transpose Code Example Part 3
L5-5.13-Transpose Code Example Part 4



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





L5-5.15-Theoretical Peak Bandwidth

From deviceQuery:

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 deviceQuery:

Memory clock: 2508×10^6 clocks/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

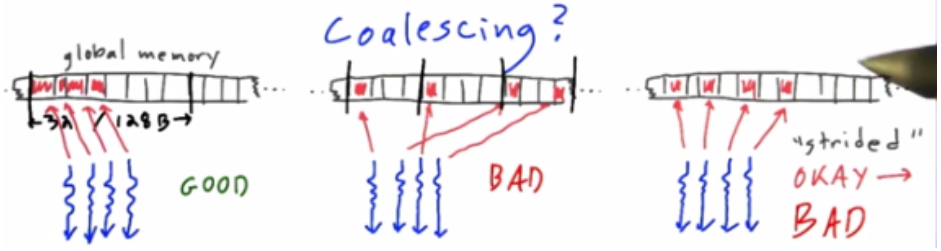
L5-5.17-Measuring Memory Usage of Our Transpose Code

Transpose - code example

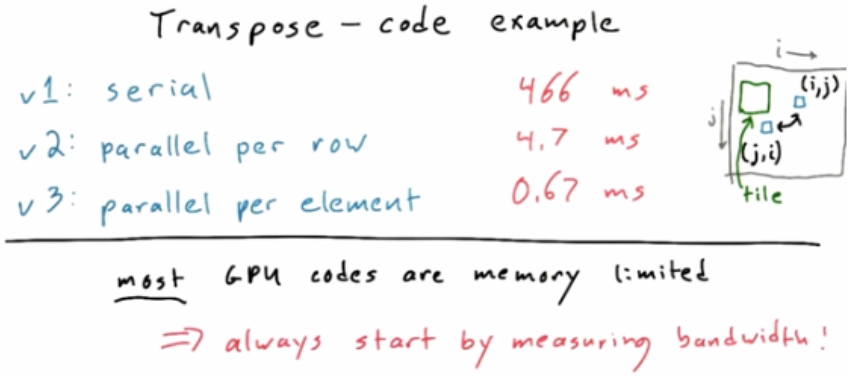
v1: serial	466 ms	0.018 GB/s
v2: parallel per row	4.7 ms	1.8 GB/s
v3: parallel per element	0.67 ms	12.5 GB/s

Transpose - code example

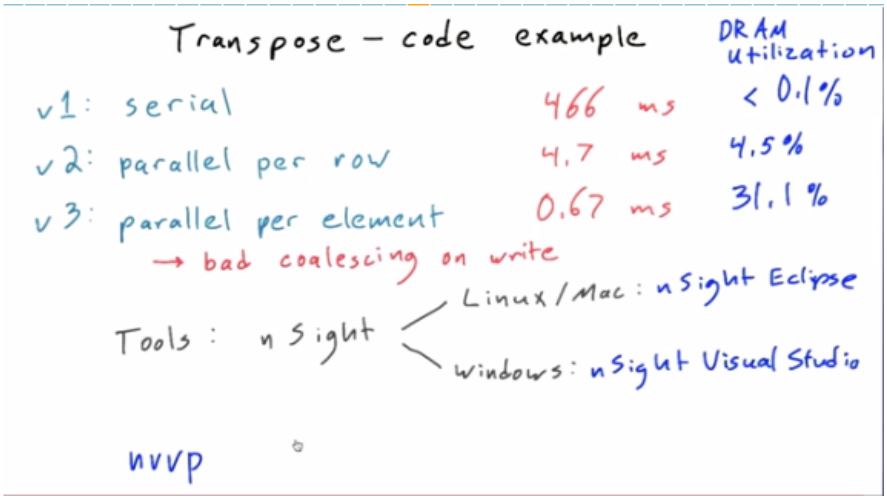
		DRAM utilization
v1: serial	466 ms	< 0.1%
v2: parallel per row	4.7 ms	4.5%
v3: parallel per element	0.67 ms	31.1%



L5-5.18-Most GPu Programs Are Memory Limited

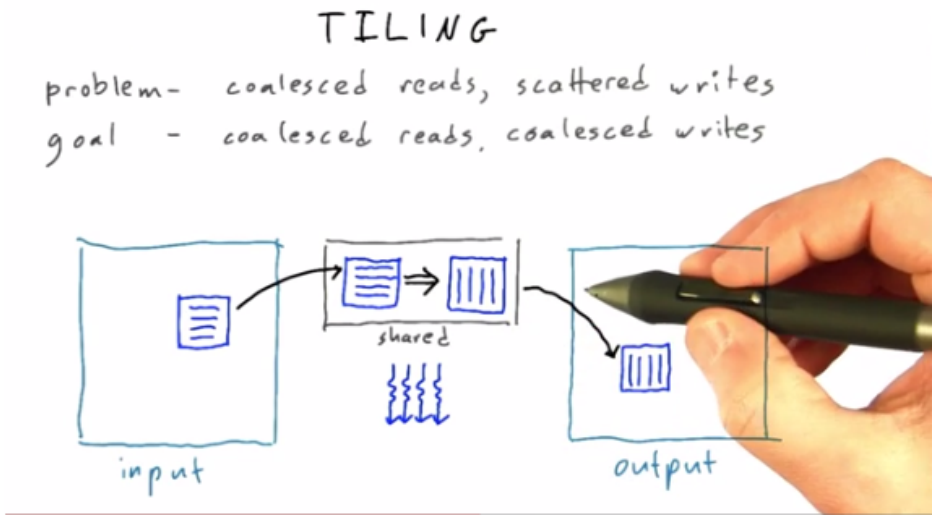


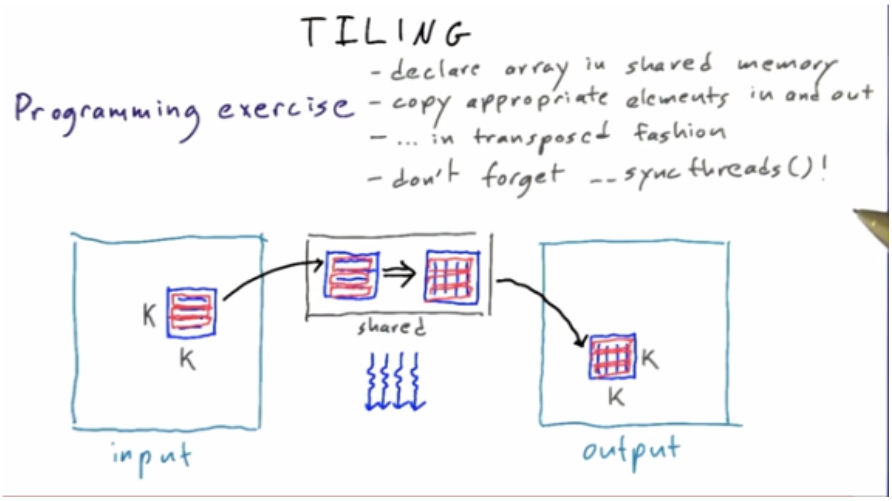
L5-5.19-Tools to Measure Bandwidth Utilization



L5-5.20-Using NVVP Part 1
L5-5.21-Using NVVP Part 2

L5-5.22-Tiling

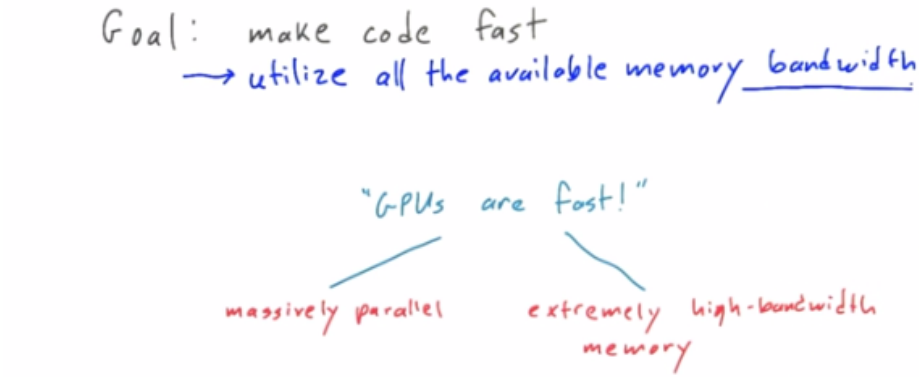


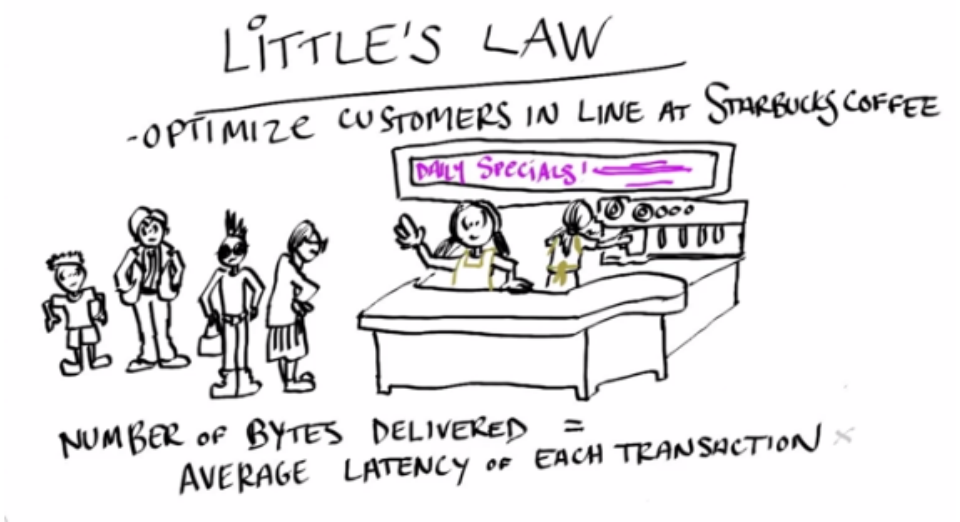


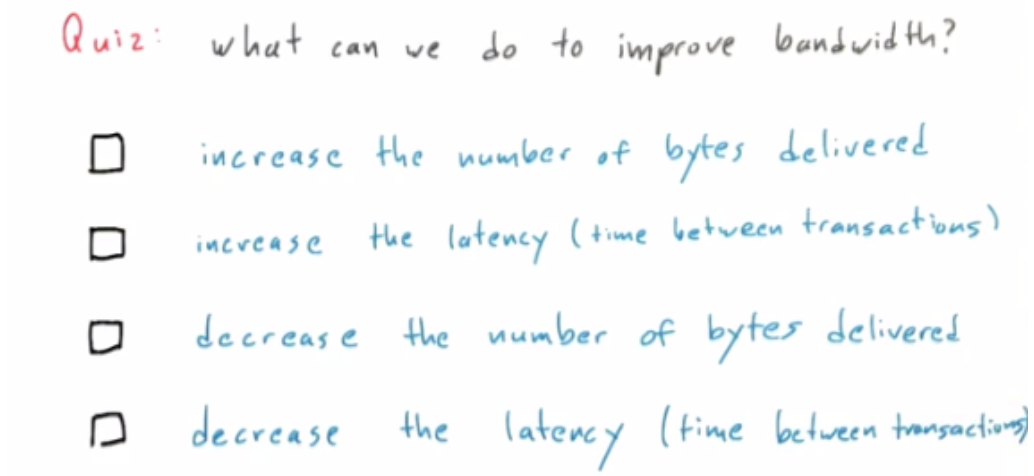
L5-5.23-Profiling the Tiling Code

NVVP analysis

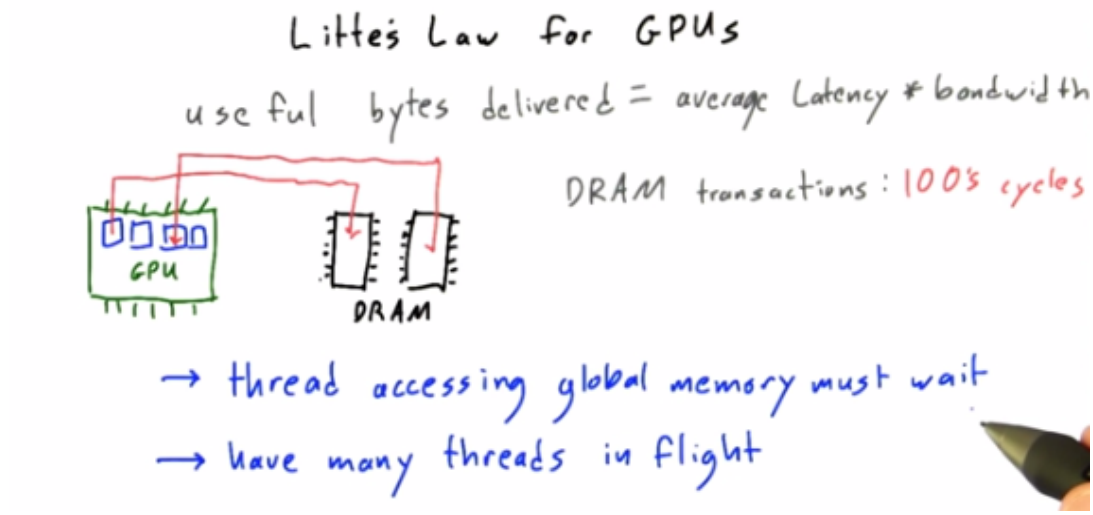
L5-5.24-Littles Law



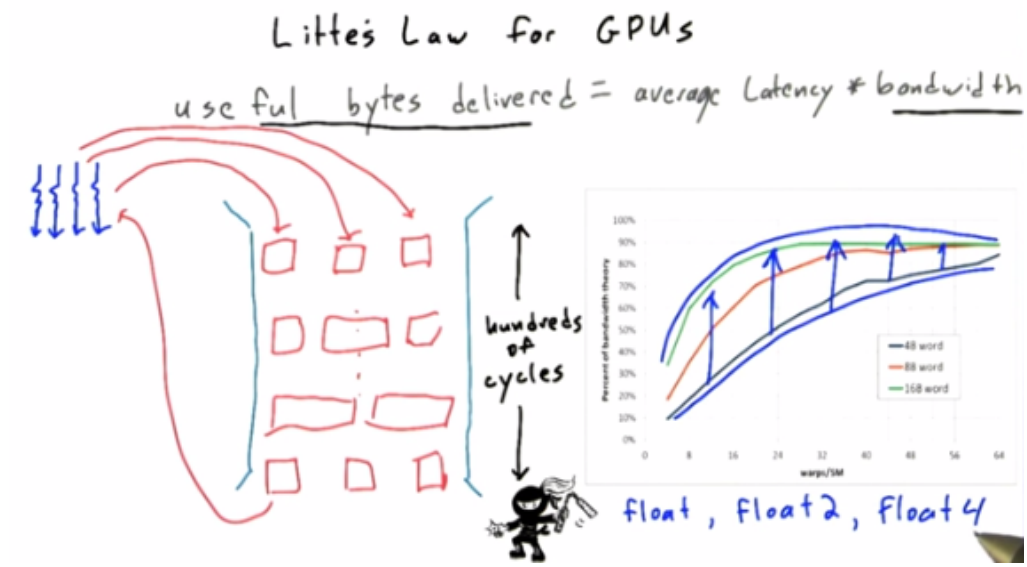


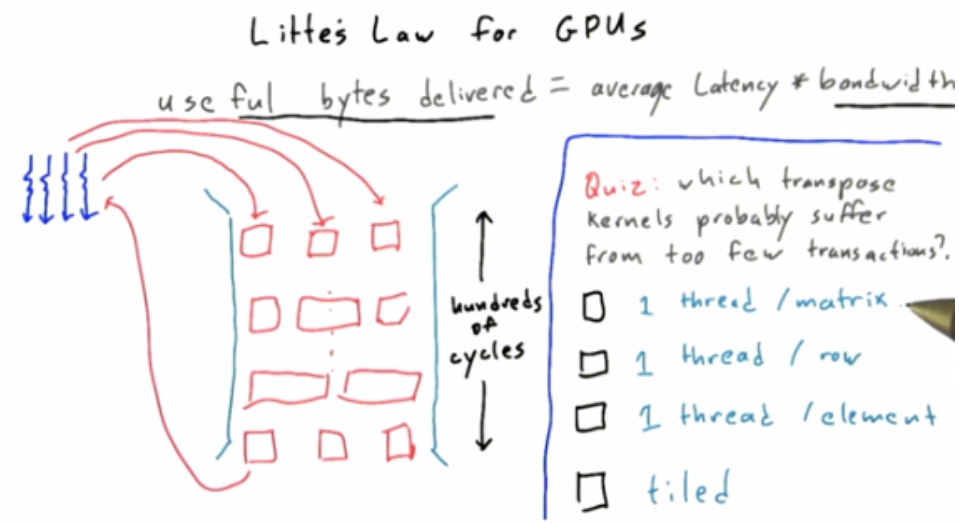


L5-5.25-Littles Law for GPUs

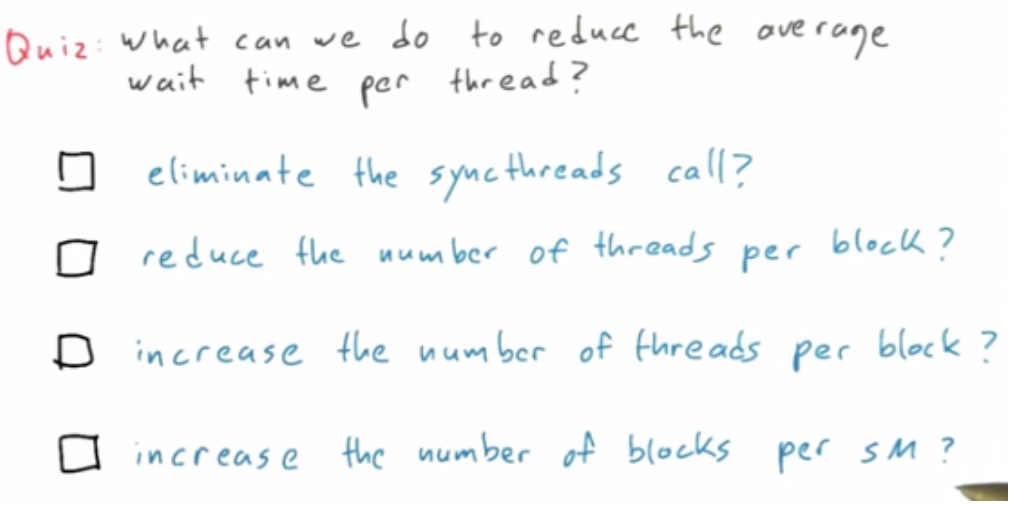


L5-5.26-Useful Bytes Delivered

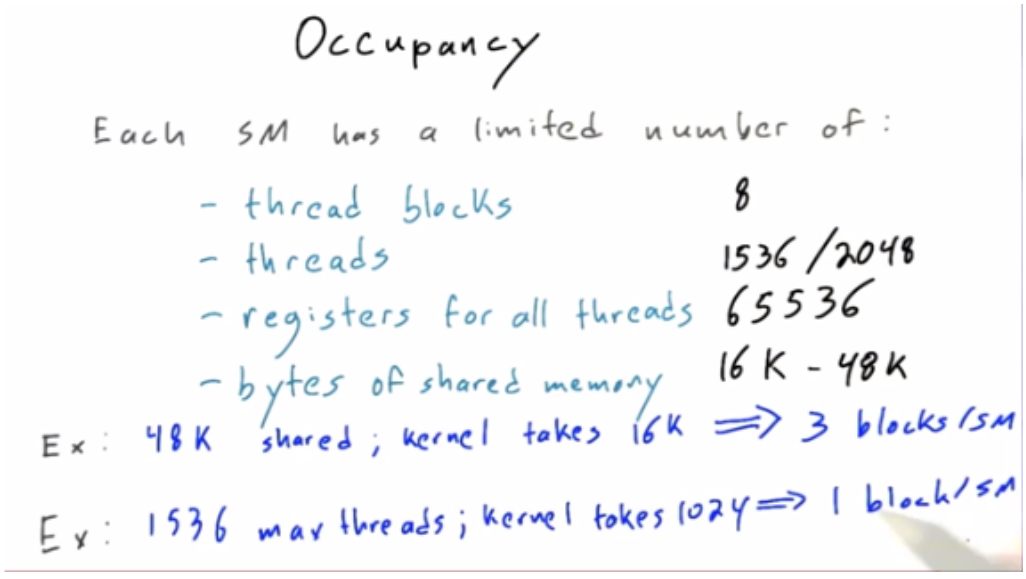




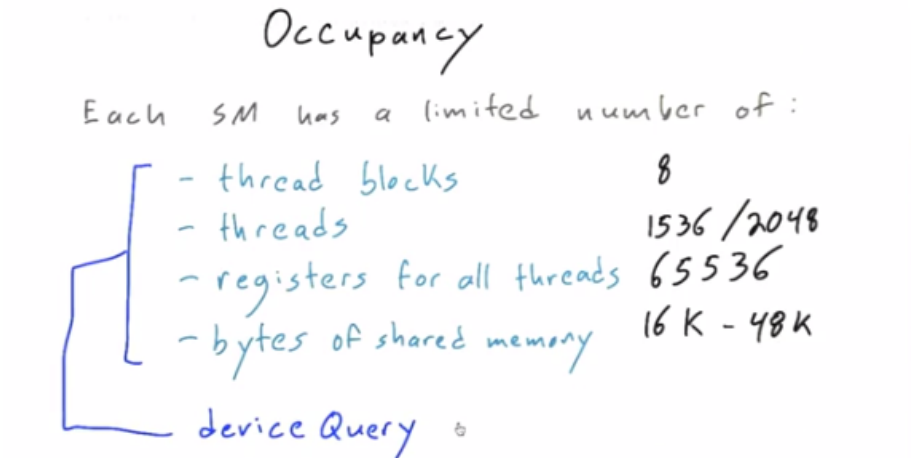
L5-5.27-Reducing Average Wait Time Per Thread

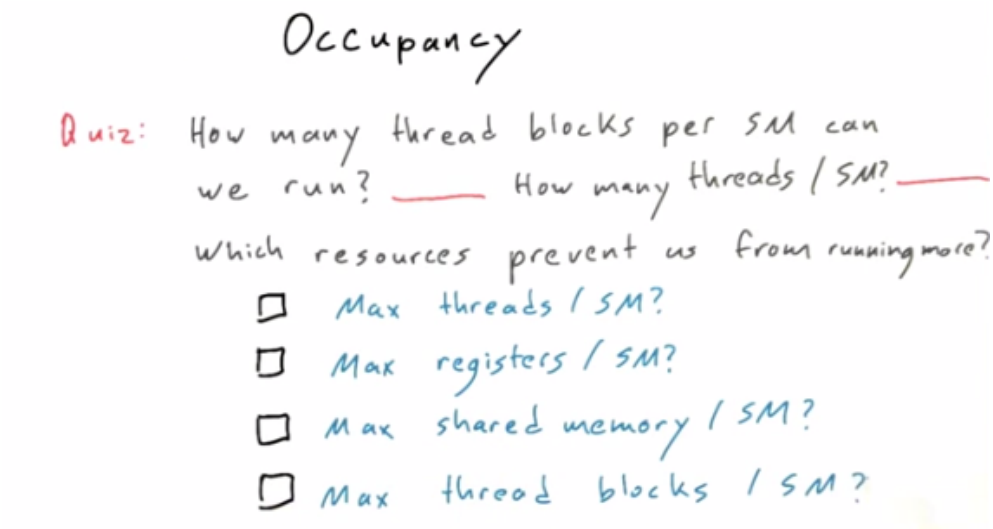


L5-5.28-Occupancy Part 1

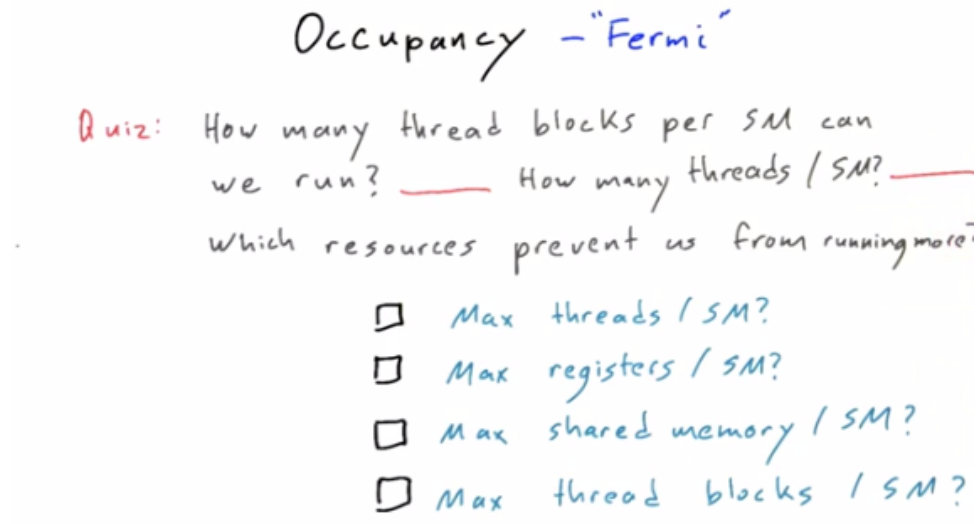


L5-5.29-Occupancy Part 2

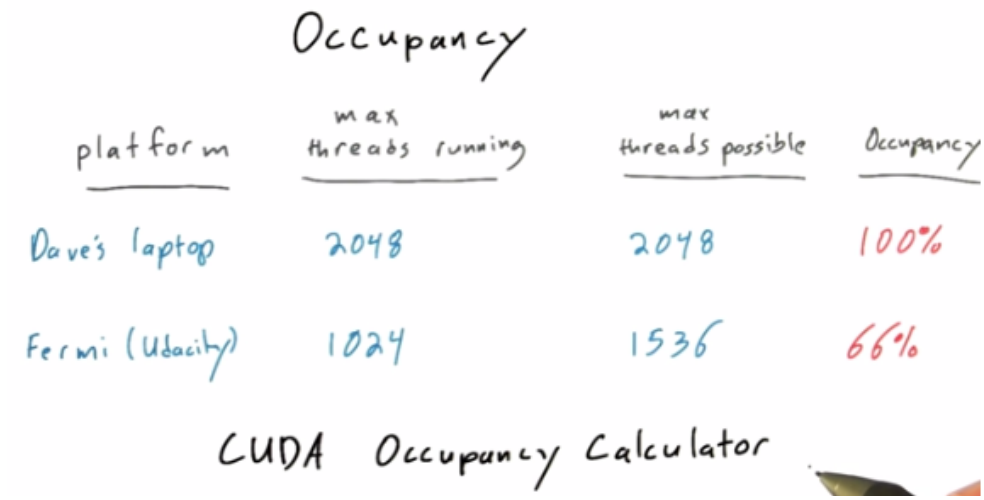




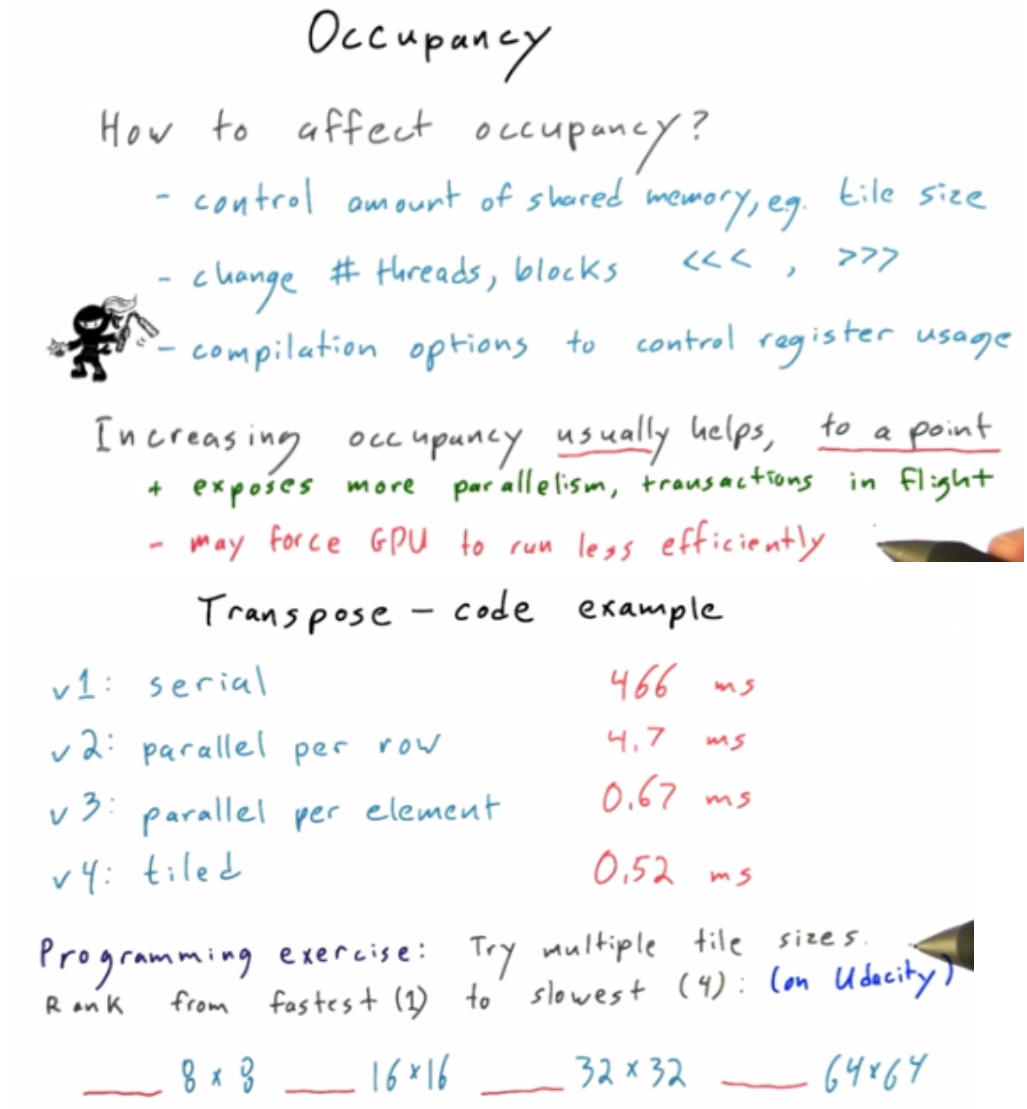
L5-5.30-Occupancy on Fermi GPUs



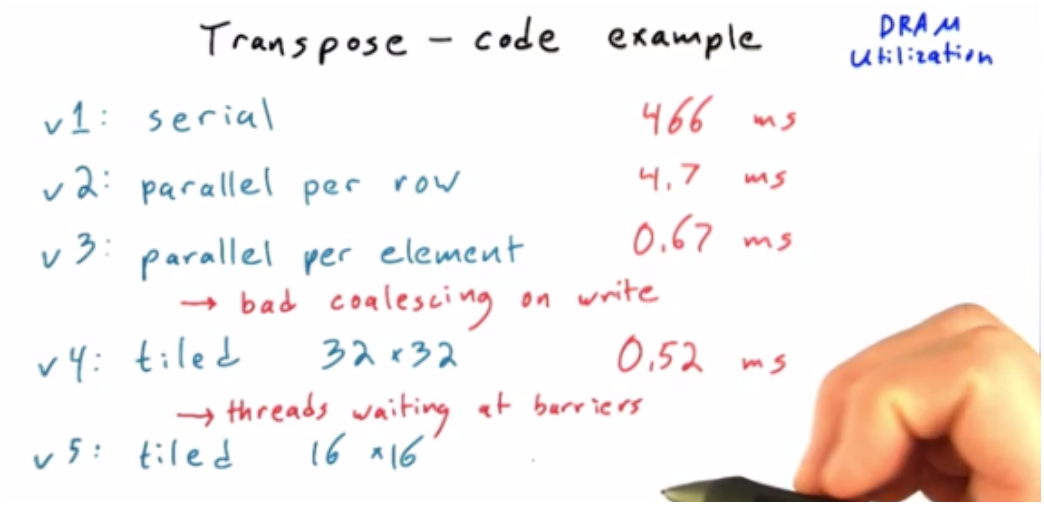
L5-5.31-Occupancy on Daves Laptop and Udacity Servers



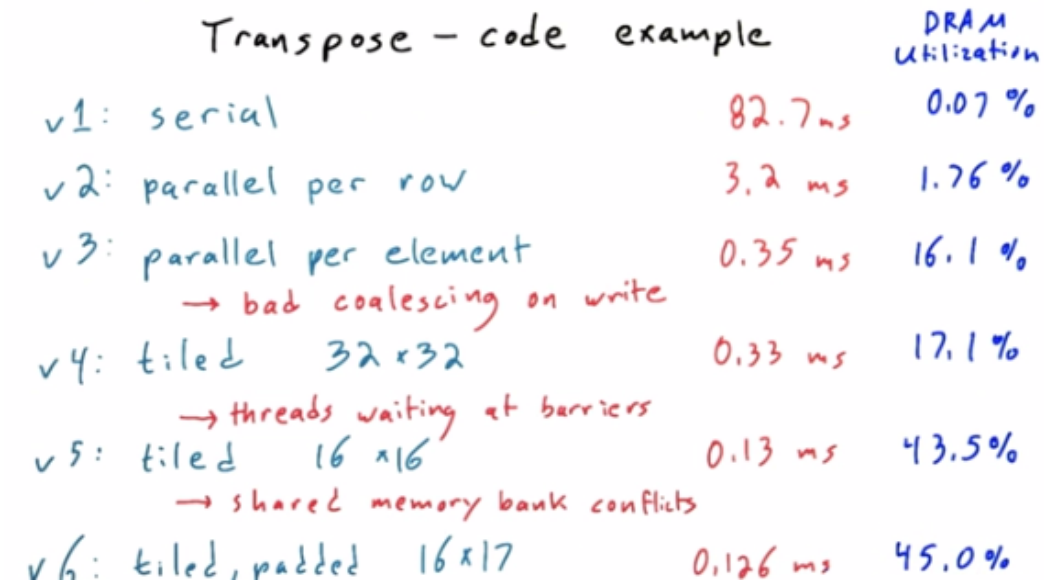
L5-5.32-How to Affect Occupancy



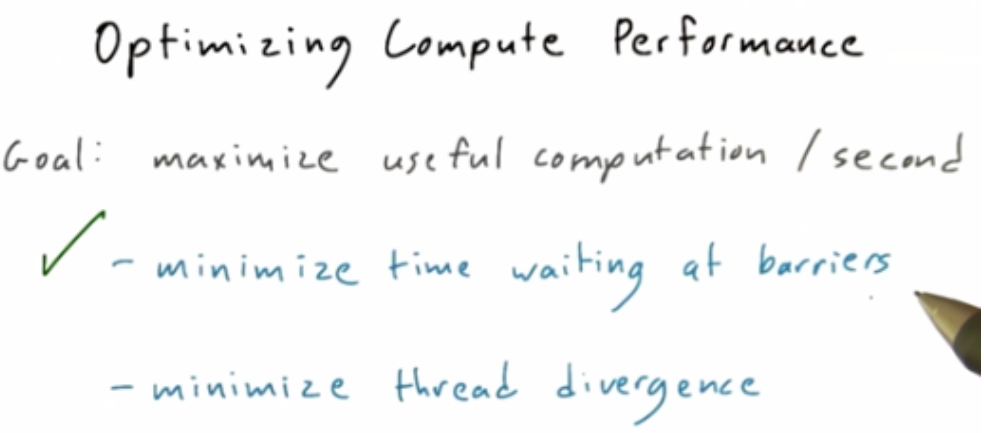
L5-5.33-Transpose Code Recap



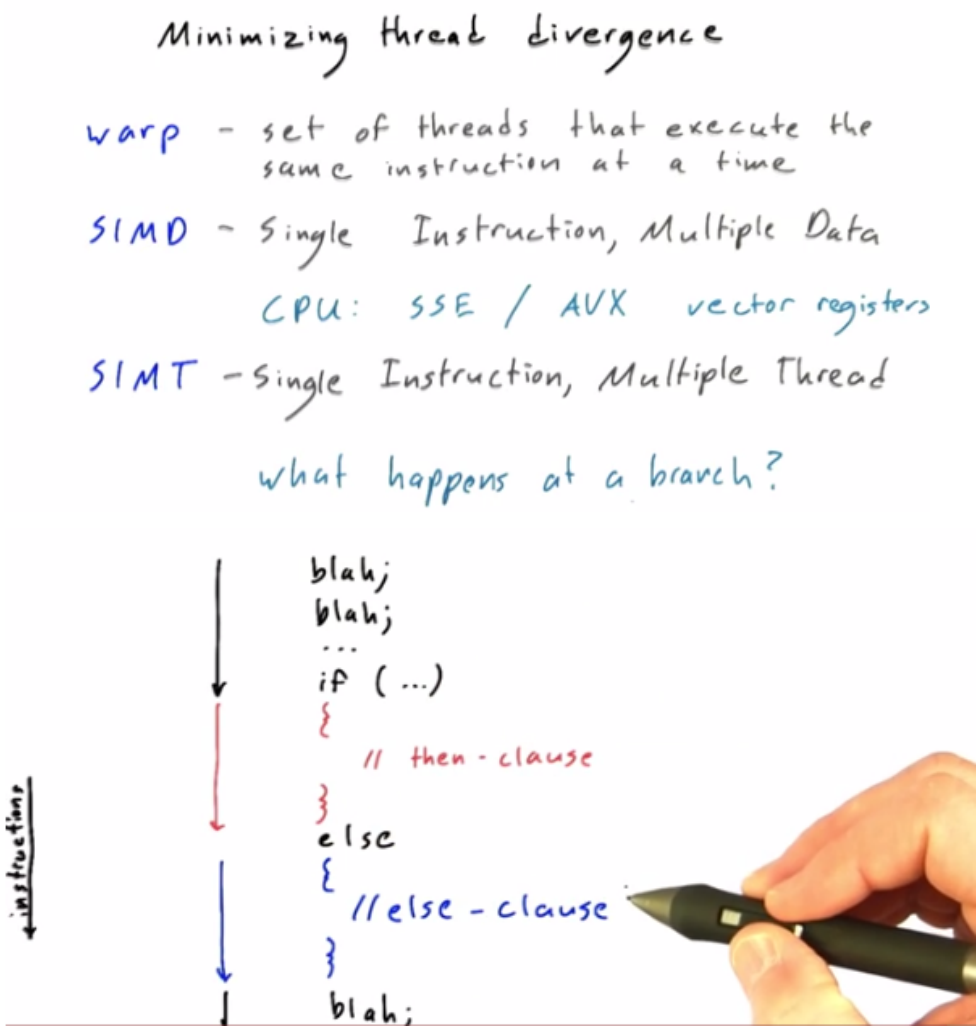
L5-5.34-Transpose Code Memory Utilization



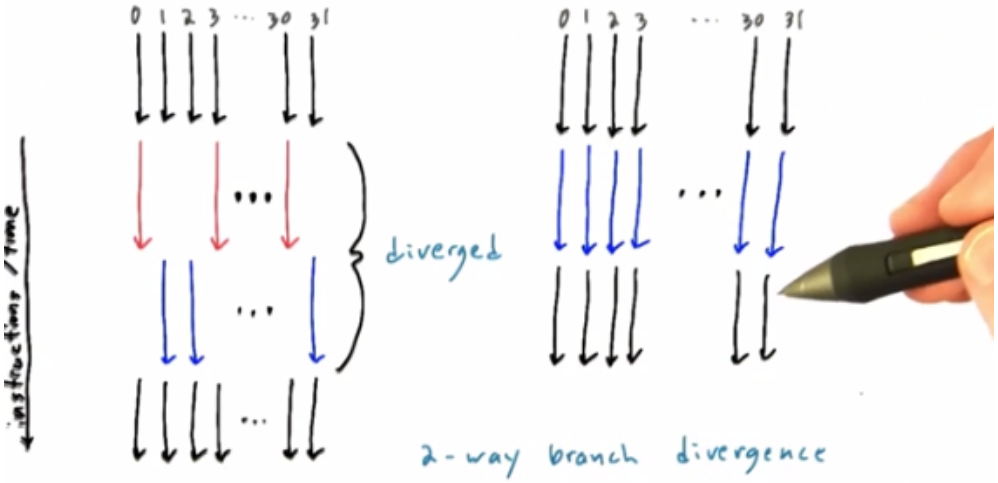
L5-5.35-Optimizing Compute Performance

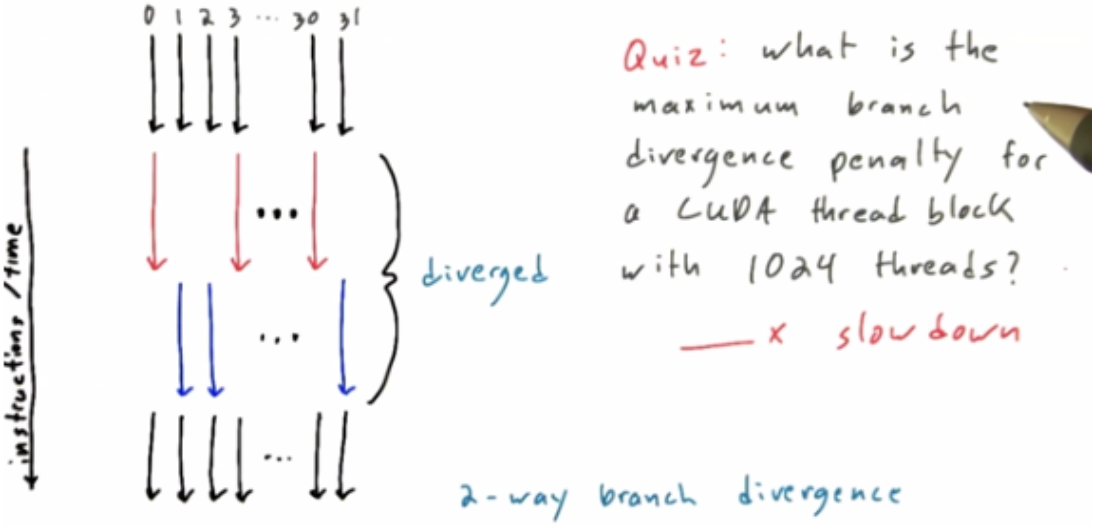


L5-5.36-Minimize Thread Divergence



L5-5.37-Thread Divergence Penalty





L5-5.38-Switch Statements and Thread Divergence Part 1

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

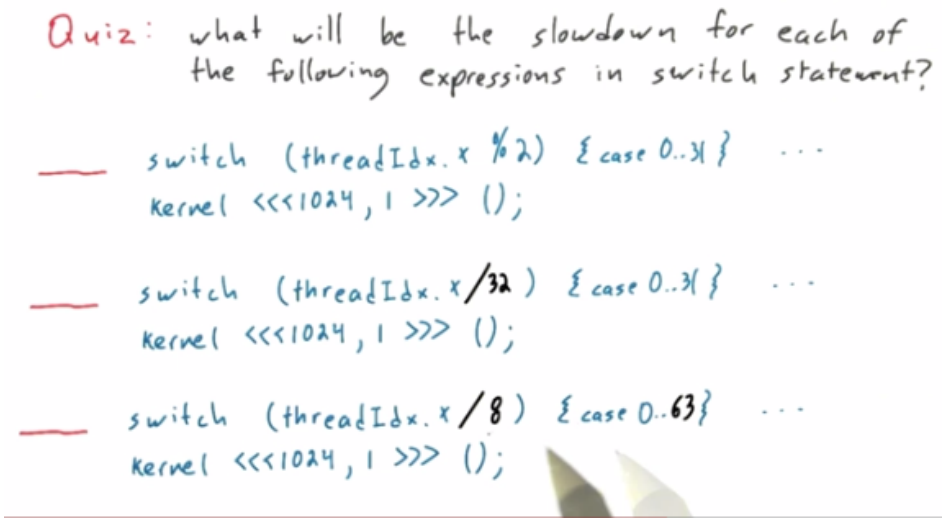
32 switch (threadIdx.x % 32) { case 0..31} ...
Kernel <<< 1024, 1 >>> ();

___ switch (threadIdx.x % 64) { case 0..63} ...
Kernel <<< 1024, 1 >>> ();

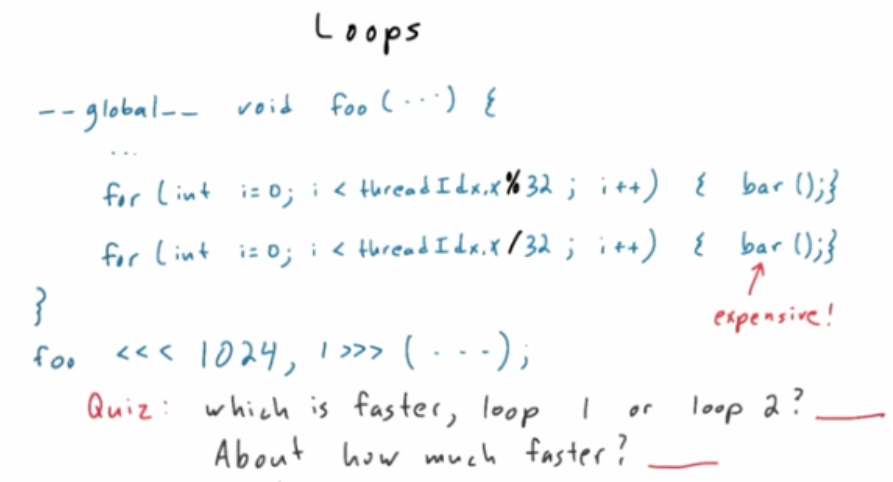
___ switch (threadIdx.y) { case 0..31} ...
Kernel <<< 64x16, 1 >>> ();

___ switch (threadIdx.y) { case 0..31} ...
Kernel <<< 16x16, 1 >>> ();

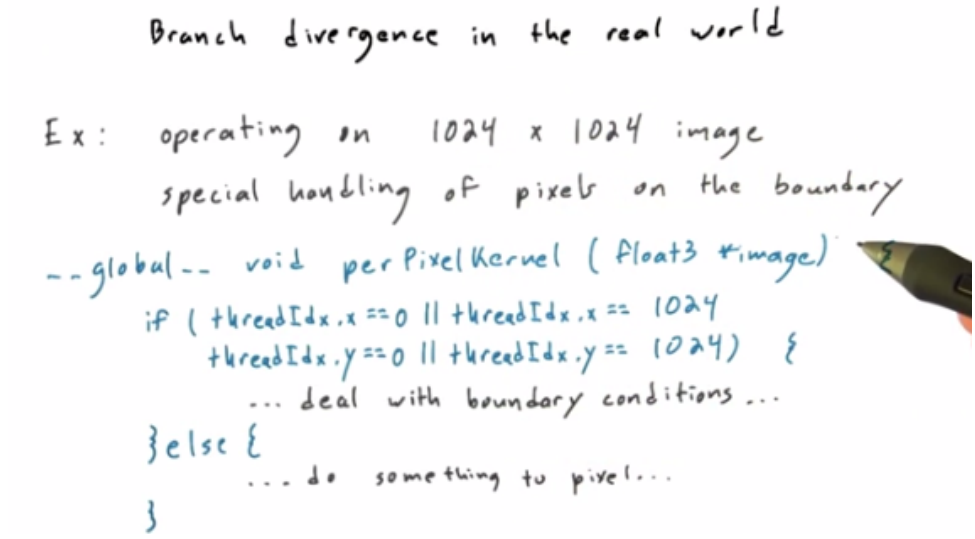
L5-5.39-Switch Statements and Thread Divergence Part 2

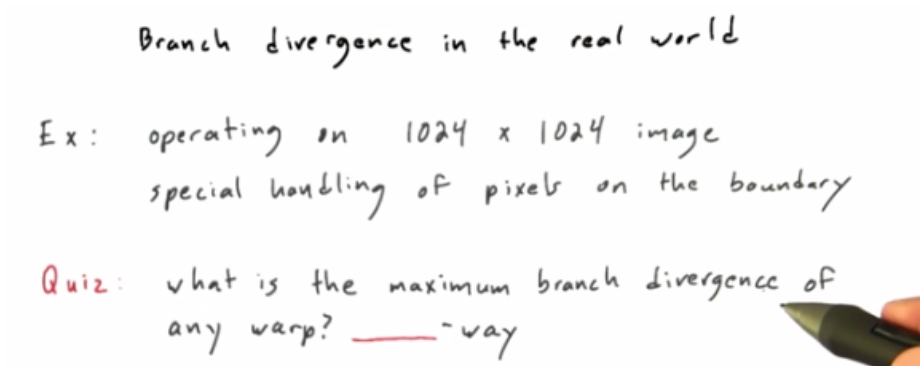


L5-5.40-Loops and Thread Divergence

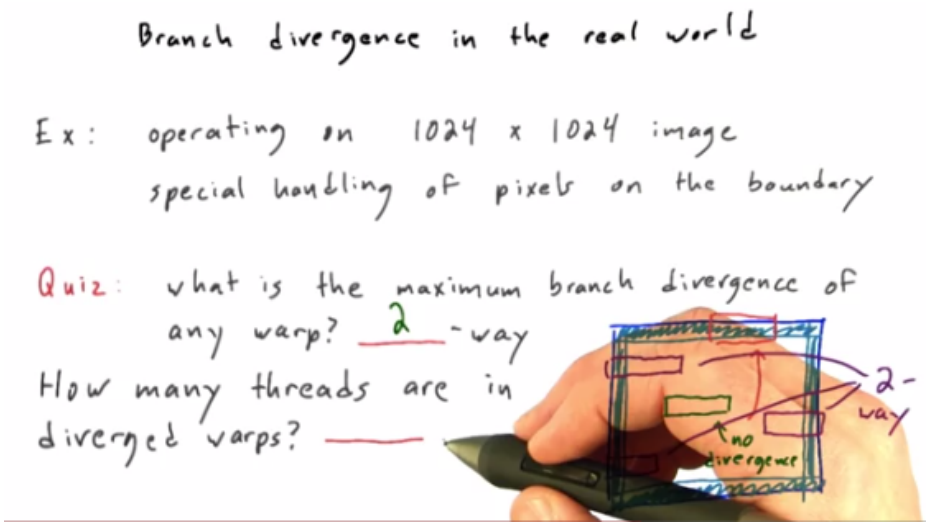


L5-5.41-Thread Divergence in the Real World Part 1

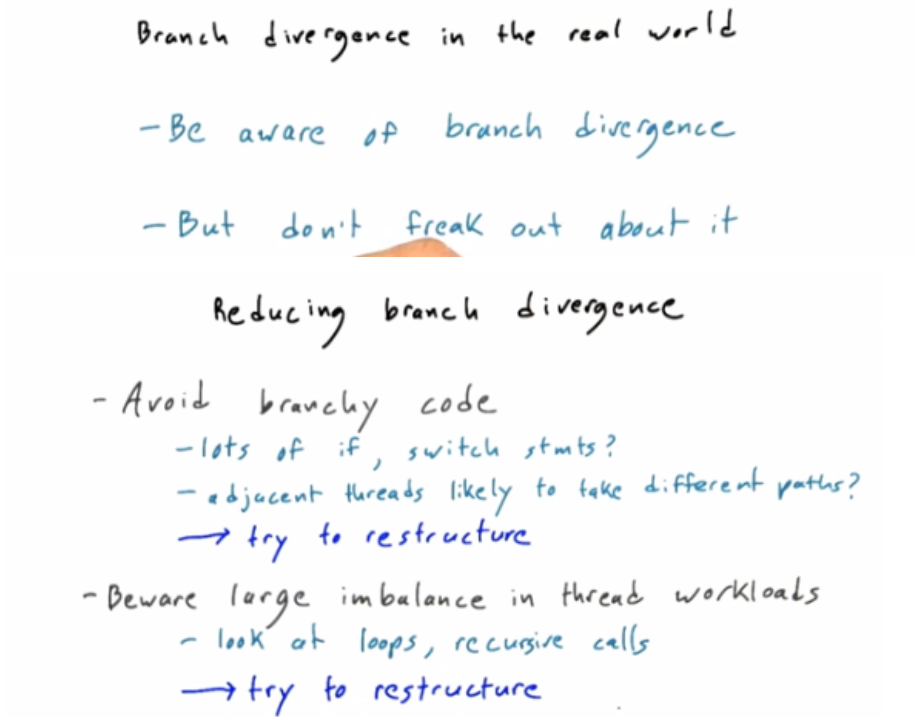




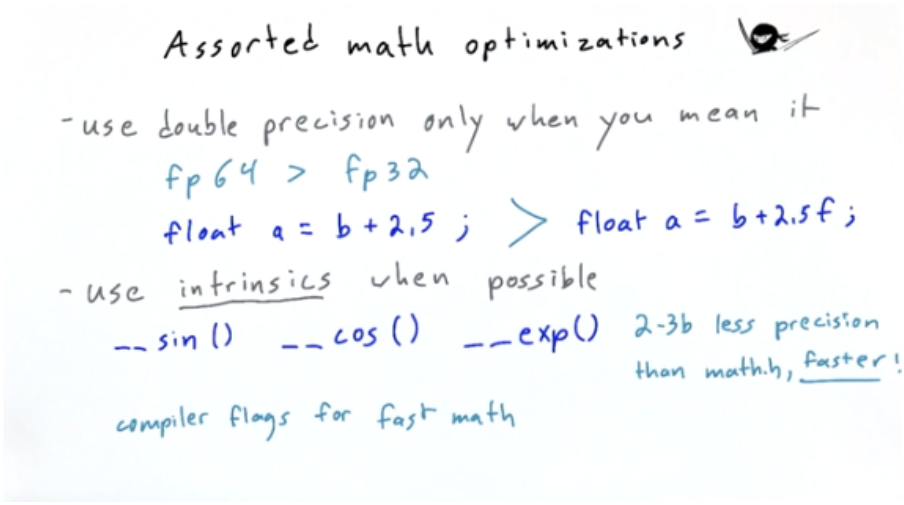
L5-5.42-Thread Divergence in the Real World Part 2



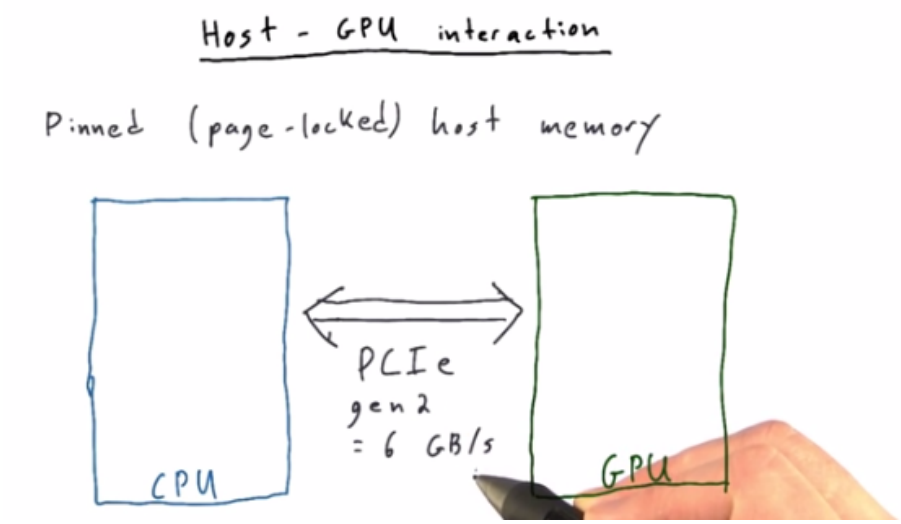
L5-5.43-Thread Divergence in the Real World Part 3

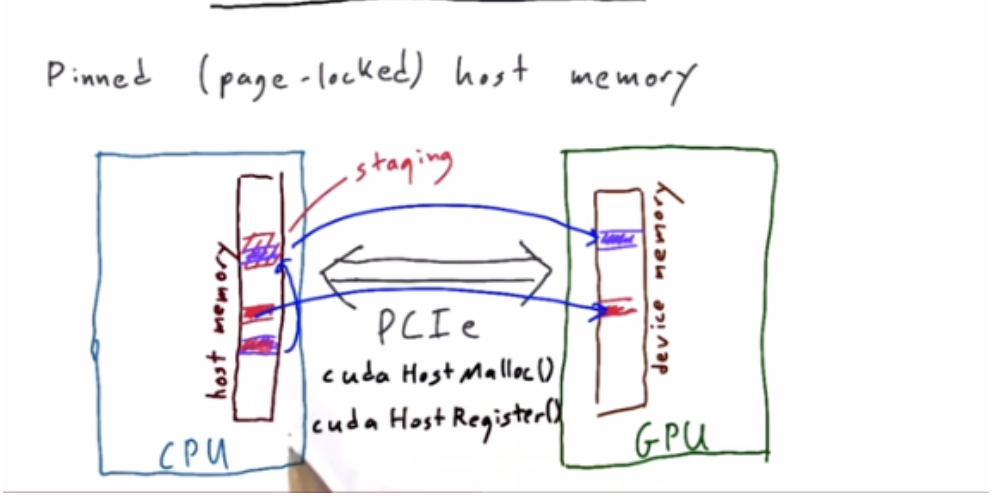


L5-5.44-Assorted Math Optimizations



L5-5.45-Host-GPU Interaction





Host - GPU interaction

Pinned (page-locked) host memory

- faster!
- enables cudaMemcpyAsync();

cudaMemcpy(...); — blocks until transfer completes

cudaMemcpyAsync(...); — execution continues immediately

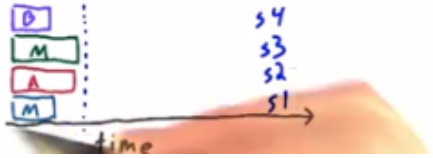
L5-5.46-Streams Part 1

Streams

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

```
cudaMemcpyAsync(...);
A<<<...>>>(...);
cudaMemcpyAsync(...);
B<<<...>>>(...);
```


```
cudaMemcpyAsync(..., s1);
A<<<...>>>(..., s2);
cudaMemcpyAsync(..., s3);
B<<<...>>>(..., s4);
```



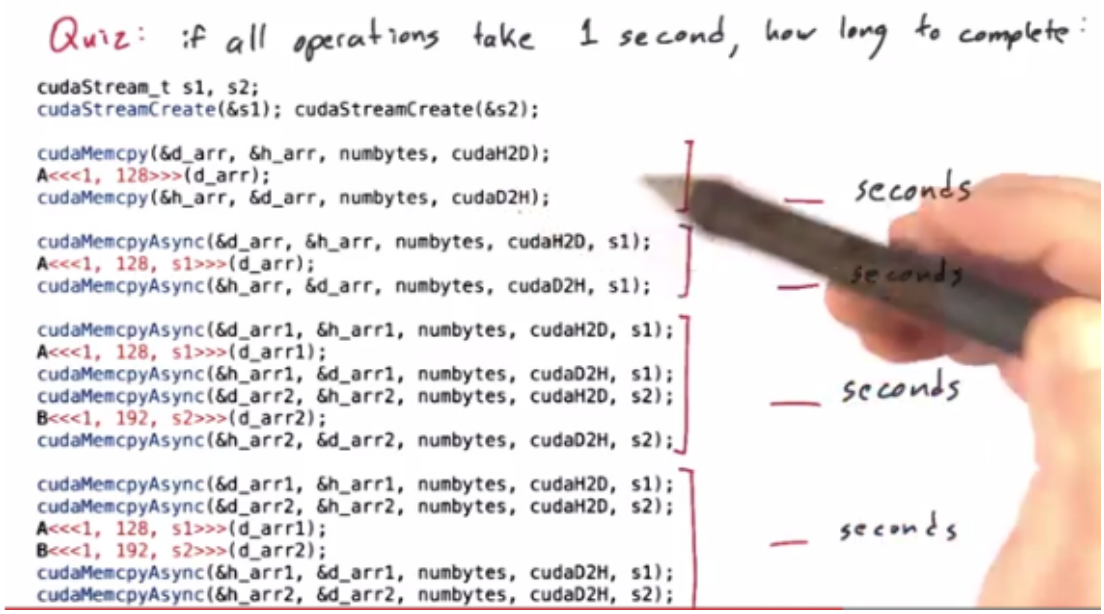
Streams

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

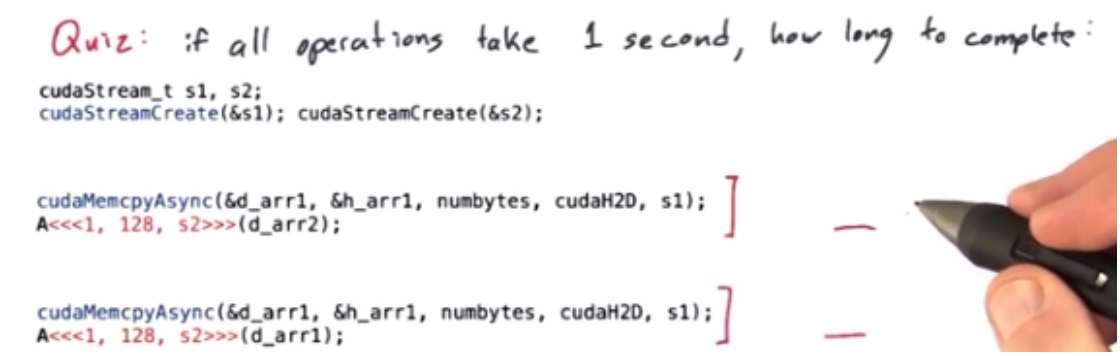
```
cudaStream_t s1;
cudaStreamCreate(&s1);
cudaStreamDestroy(s1);
```

asynchronous memory transfers → cudaMemcpyAsync → pinned memory

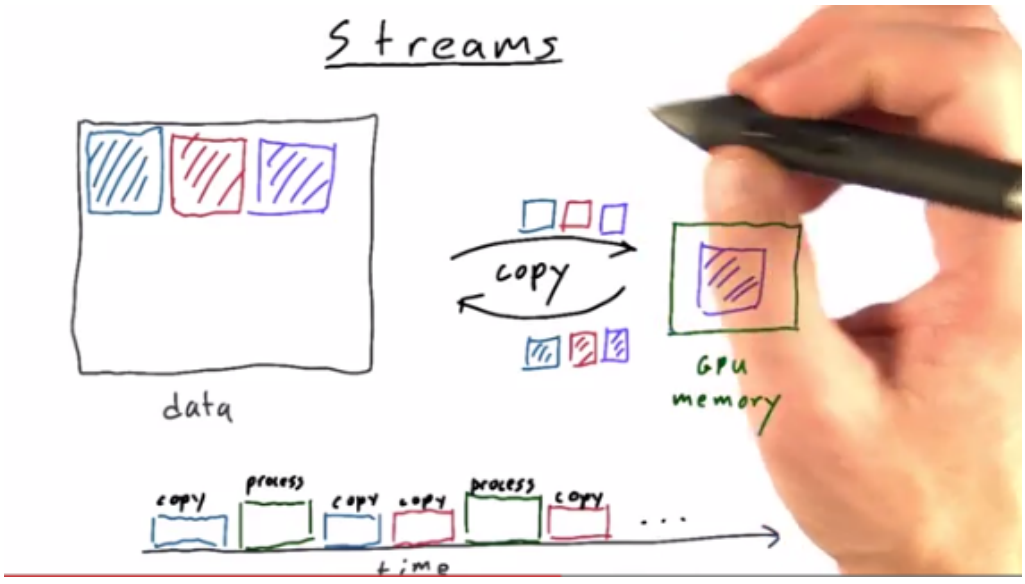
L5-5.47-Streams Part 2

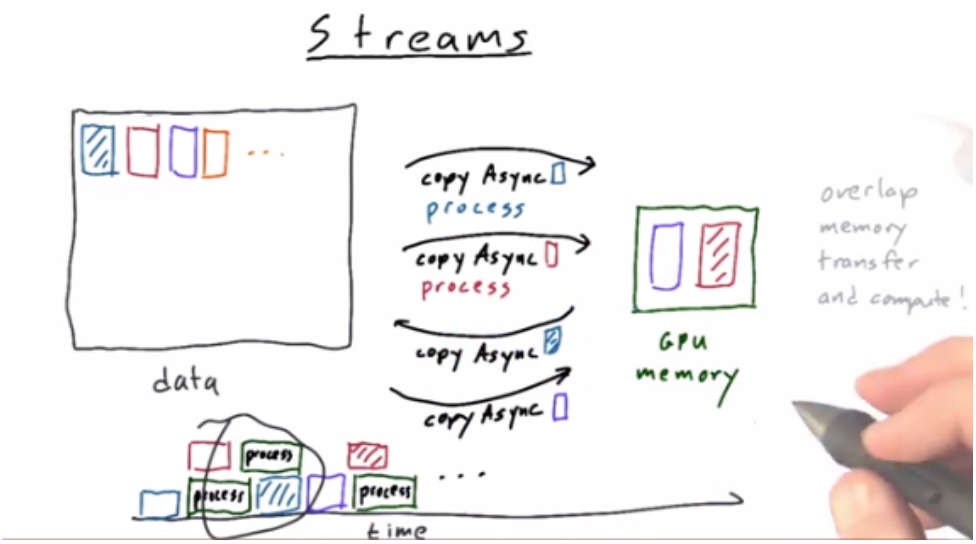


L5-5.48-Streams Part 3

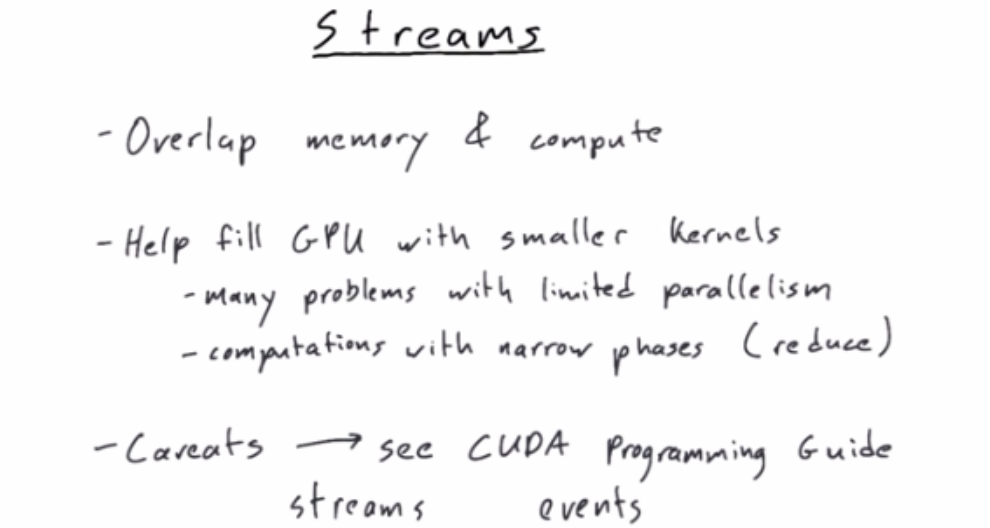


L5-5.49-Streams are Useful For

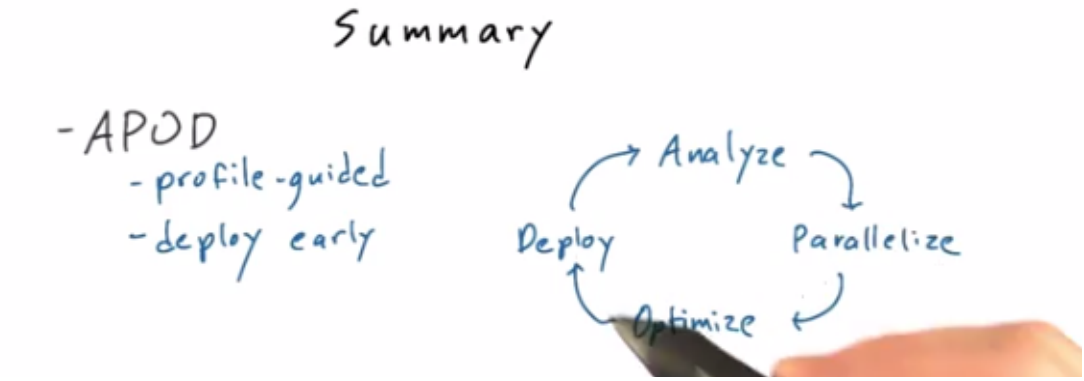




L5-5.50-Advantage of Streams



L5-5.51-Summary



Summary

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

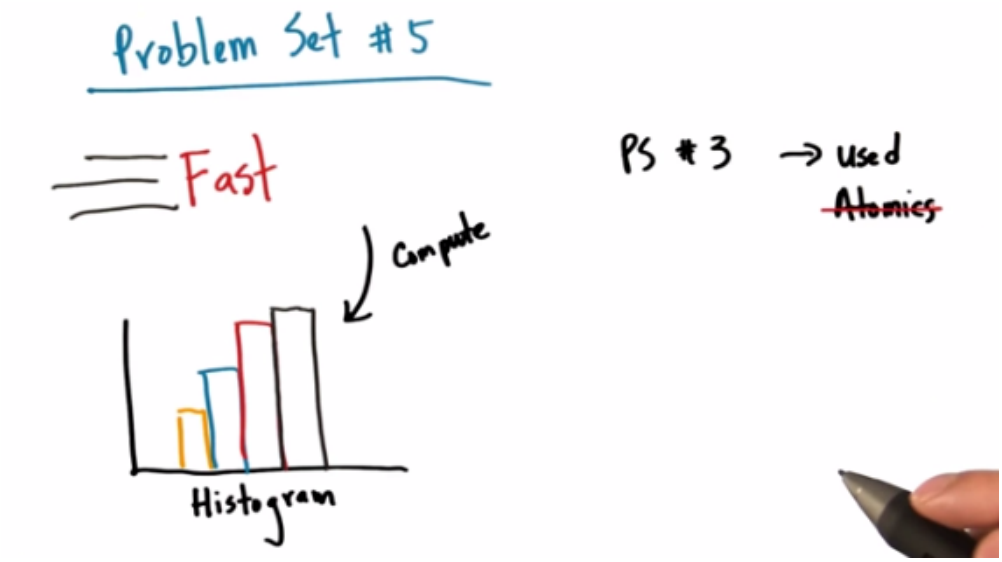
Summary

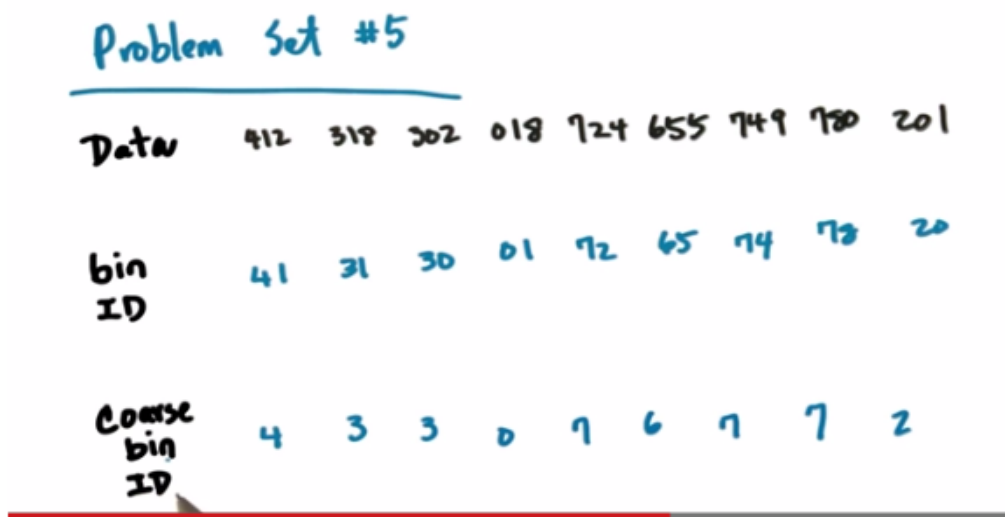
- APOD
- measure & improve memory bandwidth
- minimize thread divergence
- consider fast math
  - intrinsics __sin(), __cos(), etc
  - use double precision on purpose: 3.14 ≠ 3.14f

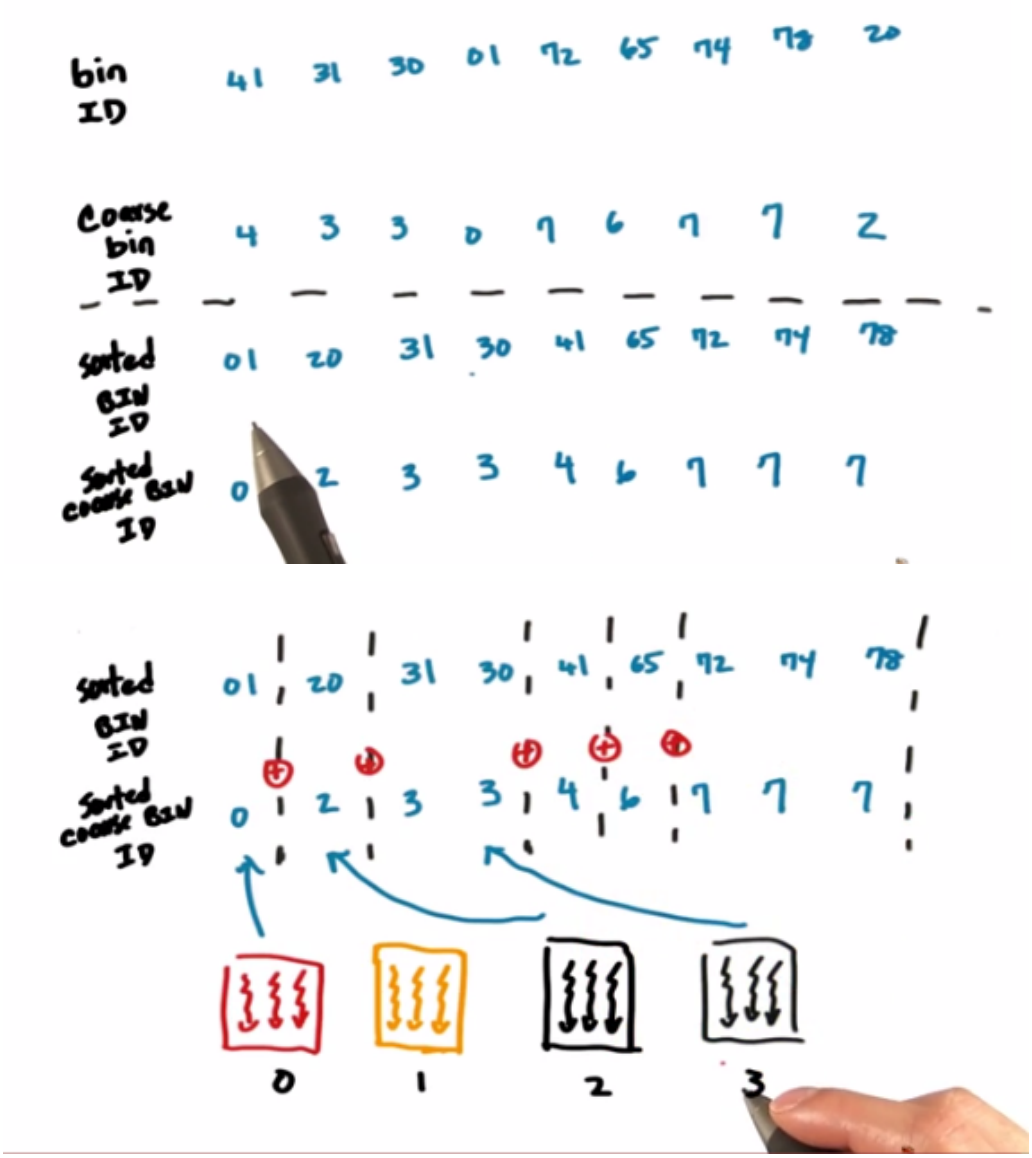
Summary

- 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






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