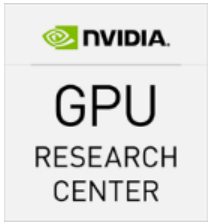
Parallel Computing

Dr Paul Ric
http://paulrichmond.shef.ac COM4521/



Assignment Feedback

Last Week

- We learnt about warp level CUDA
- How threads are scheduled and executed
  - Impacts of divergence
- Atomics: Good and bad…
- Do the warp shuffle!
- Parallel primitives
- Scan and Reduction

Credits

- The code and much of the content from this lecture is based on the GTC2016 Talk by C. Angerer and J. Progsch (NVIDIA)
  - S6112 – CUDA Optimisation with NVIDIA Nsight for Visual Studio
  - Provided by NVIDIA with thanks to Joe Bungo
- Content has been ada filer Guided Analysis where possible
- Additional steps and analysis have d

Learning Objectives

- Understand the key performance metrics of GPU code.
- Understand profiling metrics and relate this to approaches which they have already learnt to address limiting factors in their code.
- Appreciate memory vs compute bound code and be able to recognise factors whic

- Profiling Introduction
- The Problem
- Visual Profiler Guided Analysis
  - Iteration 1
  - Iteration 2
  - Iteration 3
  - Iteration 4

The APOD Cycle

4. Deploy and Test

1. Assess
   - Identify Performance Limiter
   - Analyze Profile
   - Find Indicators

3. Optimize

3b. Build Knowledge

2. Parallelize

https://devblogs.nvidia.com/assess-parallelize-optimize-deploy/

CUDA Profiling Options

- **Visual Profiler (1st choice)**
  - Stand alone cross platform (java on Eclipse) program
  - Guided performance analysis
  - Links to CUDA best practice guide
- NVProf
  - Command line profiler
  - Results can be visualised in Visual Pro
- Visual Studio Nsight Profiler
  - Built into visual studio
  - Detailed kernel and source level analysis (more than Visual Profiler)
  - Unguided

Changes to your code

- If you want to associate profile information with source line
  - `--lineinfo` argument
  - Works in release mode
- Must flush GPU buffers
  - `cudaDeviceReset`
  - At end of program

Conveyor belt model

- Our GPU program is like a factory assembly line
  - Data in and data out (in a new form)
  - Skilled operators (multi processors) doing stuff with chunks of the data
  - Both the belt and people have maximum operating speed

- Ideal situation
  - Conveyor belt runs at full speed
  - Skilled operators always 100% busy

What is likely to effect this model?

Potential Performance Limiters

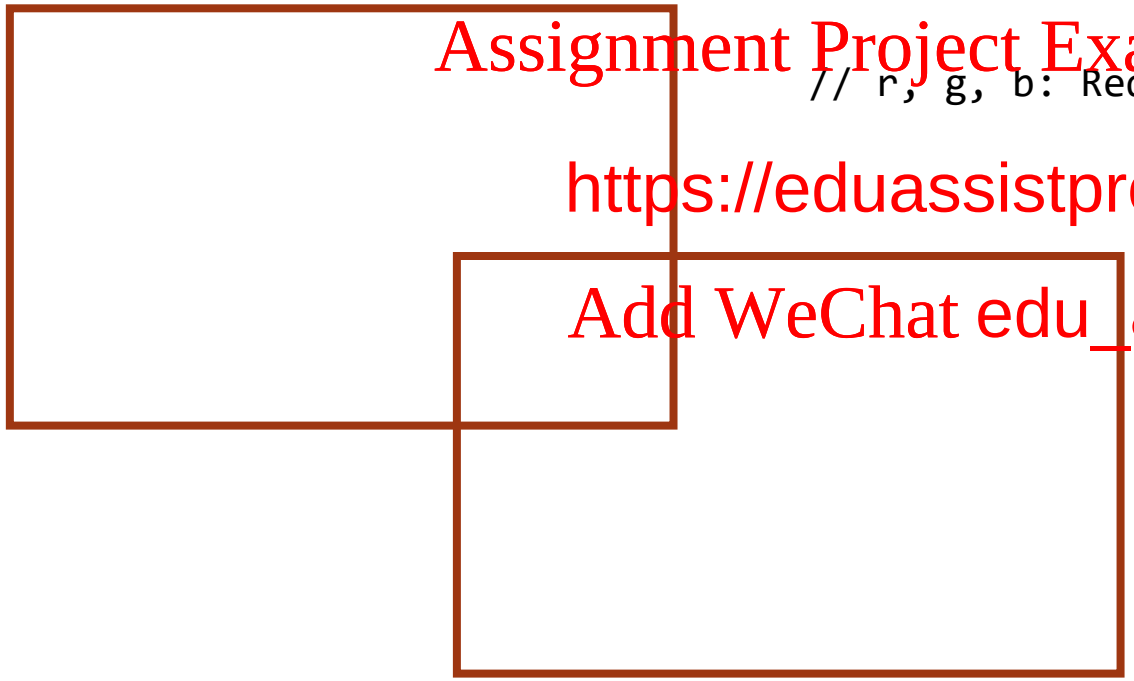
- Memory
  - Program limited by memory bandwidth
  - Can't get data to the registers on the device fast enough
  - *Are you using lots of global memory but not faster local memory caches?*
  - *Have you exceed the amount of cache available?*
- Compute
  - Memory bandwidth well
  - GPU is too busy perform
  - *Have you got high levels of divergence wit* *xecution efficiency ?*
- Latency
  - Poor occupancy = not enough active threads
  - Instruction execution stalls due to poor memory access patterns (sparse or poorly used data)
  - *Is problem size or block size too small? Are you using the memory bandwidth effectively (cache line utilisation)?*

- Profiling Introduction
- The Problem
- Visual Profiler Guided Analysis
  - Iteration 1
  - Iteration 2
  - Iteration 3
  - Iteration 4

Introducing the Application

Introducing the Application

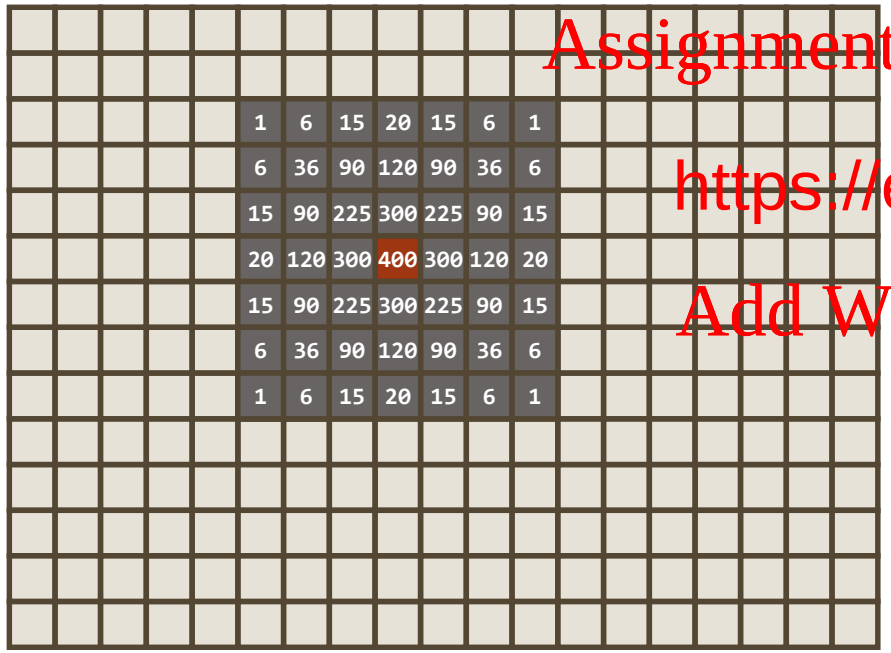
- Grayscale Conversion




```
// r, g, b: Red, green, blue components of the pixel p
                                    + 0.586811f*g + 0.114350f*b;
```

Introducing the Application

- Blur: 7x7 Gaussian Filter




```
foreach pixel p:
    d sum of p and its 48 neighbors
```

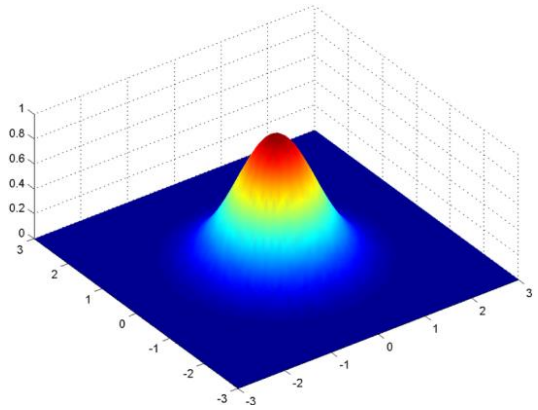


Image from Wikipedia

The University Of Sheffield.

NVIDIA GPU RESEARCH CENTER

Introducing the Application

- Edges: 3x3 Sobel Filters

```
foreach pixel p:
  Gx = weighted sum of p and its 8 neighbors
  Gy = weighted sum of p and its 8 neighbors
  p  = sqrt(Gx + Gy)
```


Gx:

-2	0	2
-1	0	1

Weights for Gy:

1	2	1
0	0	0
-1	-2	-1

The University Of Sheffield.

NVIDIA GPU RESEARCH CENTER

The Starting Code

```
void gaussian_filter_7x7_v0(int w, int h, const uchar *src, uchar *dst)
{
  // Position of the thread in the image.
  const int x = blockIdx.x*blockDim.x + threadIdx.x;
  const int y = blockIdx.y*blockDim.y + threadIdx.y;

  // Early exit if the thread is not in the image.
  if( !in_img(x, y, w, h) )
    return;

  // Load the 48 neighbours and myself.
  int n[7][7];
  for( int j = -3 ; j <= 3 ; ++j )
    for( int i = -3 ; i <= 3 ; ++i )
      n[j+3][i+3] = in_img(x+i, y+j, w, h) ? (int) src[(y+j)*w + (

  // Compute the convolution.
  int p = 0;
  for( int j = 0 ; j < 7 ; ++j )
    for( int i = 0 ; i < 7 ; ++i )
      p += gaussian_filter[j][i] * n[j][i];

  // Store the result.
  dst[y*w + x] = (uchar) (p / 256);
}
```


What is good and what is bad?

❑ https://github.com/chmaruni/nsight-gtc

The Starting Code

```
void gaussian_filter_7x7_v0(int w, int h, const uchar *src, uchar *dst)
{
  // Position of the thread in the image.
  const int x = blockIdx.x*blockDim.x + threadIdx.x;
  const int y = blockIdx.y*blockDim.y + threadIdx.y;

  // Early exit if the thread is not in the image.
  if( !in_img(x, y, w, h) )
    return;

  // Load the 48 neighbours and myself.
  int n[7][7];
  for( int j = -3 ; j <= 3 ; ++j )
    for( int i = -3 ; i <= 3 ; ++i )
      n[j+3][i+3] = in_img(x+i, y+j, w, h) ? (int) src[(y+j)*w + (

  // Compute the convolution.
  int p = 0;
  for( int j = 0 ; j < 7 ; ++j )
    for( int i = 0 ; i < 7 ; ++i )
      p += gaussian_filter[j][i] * n[j][i];

  // Store the result.
  dst[y*w + x] = (uchar) (p / 256);
}
```

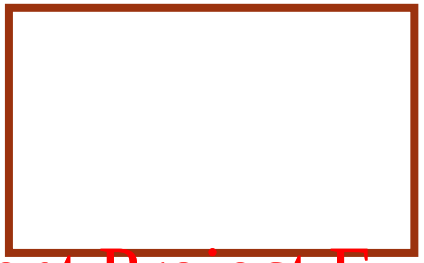

What is good and what is bad?

❑ https://github.com/chmaruni/nsight-gtc

Profiling Machine

- NVIDIA GeForce GTX980
  - GM200
  - Compute Capability SM5.2
- CUDA 7.0
- Windows 7
- Visual Studio 2013
- Nsight Visual Studio Edition 5.0

- Profiling Introduction
- The Problem
- Visual Profiler Guided Analysis
  - Iteration 1



Device Activity

Hints

Results

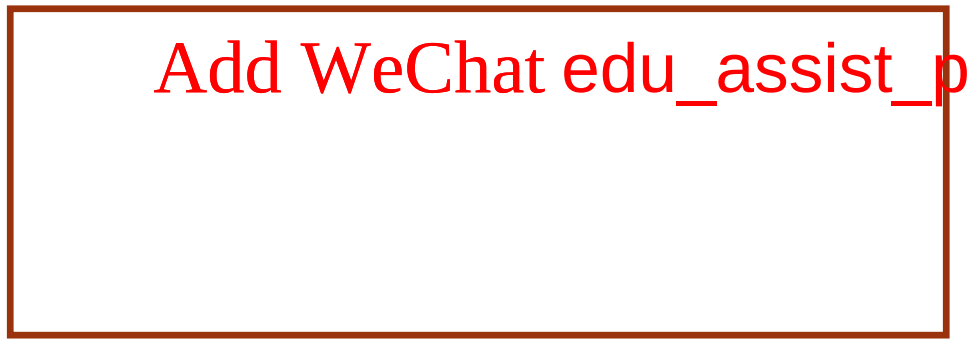
- We are using only a single stream
- Kernels have data dependencies so can't be executed in parallel

- This is a problem
- The guided analysis will try and address this

Guided Analysis

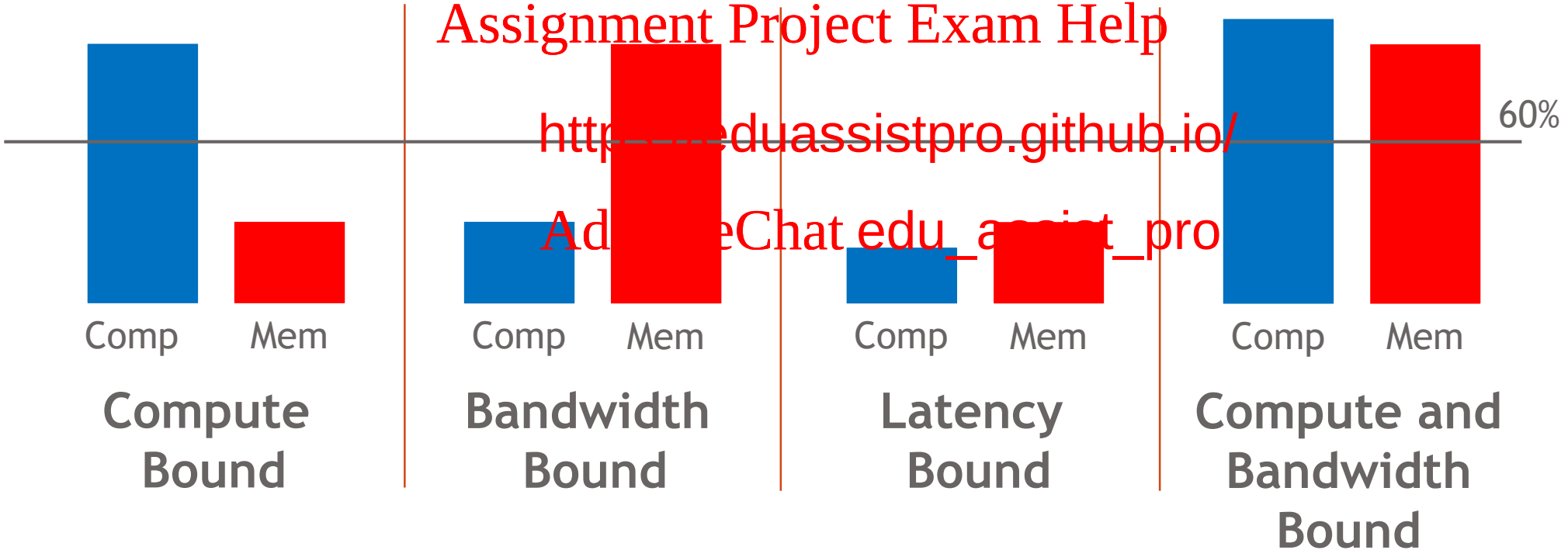
guassian_filter_7x7_v0 kernel has highest rank

What is this telling us about our code?



Memory Bound Problem!

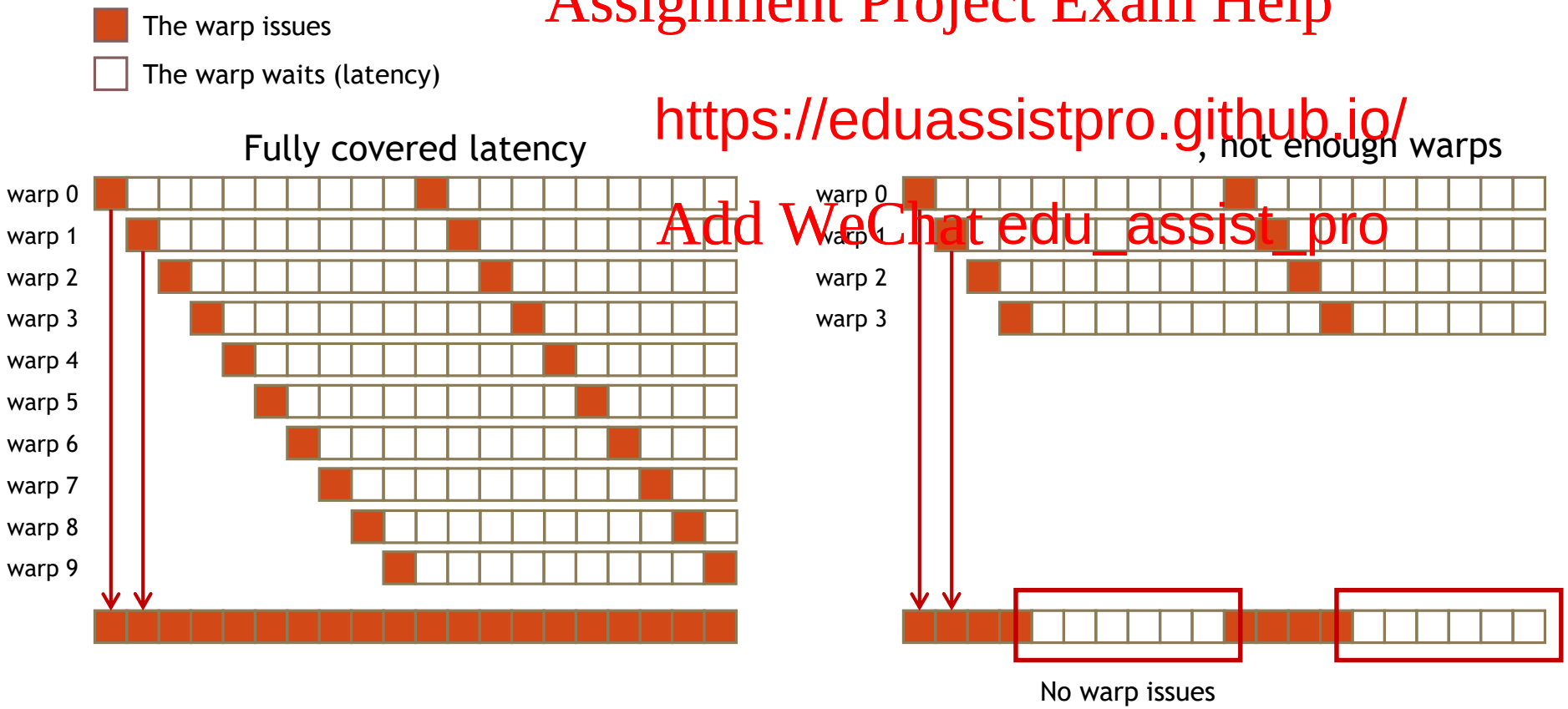
Memory vs Compute vs Latency



Better Occupancy might improve compute use

What about occupancy?

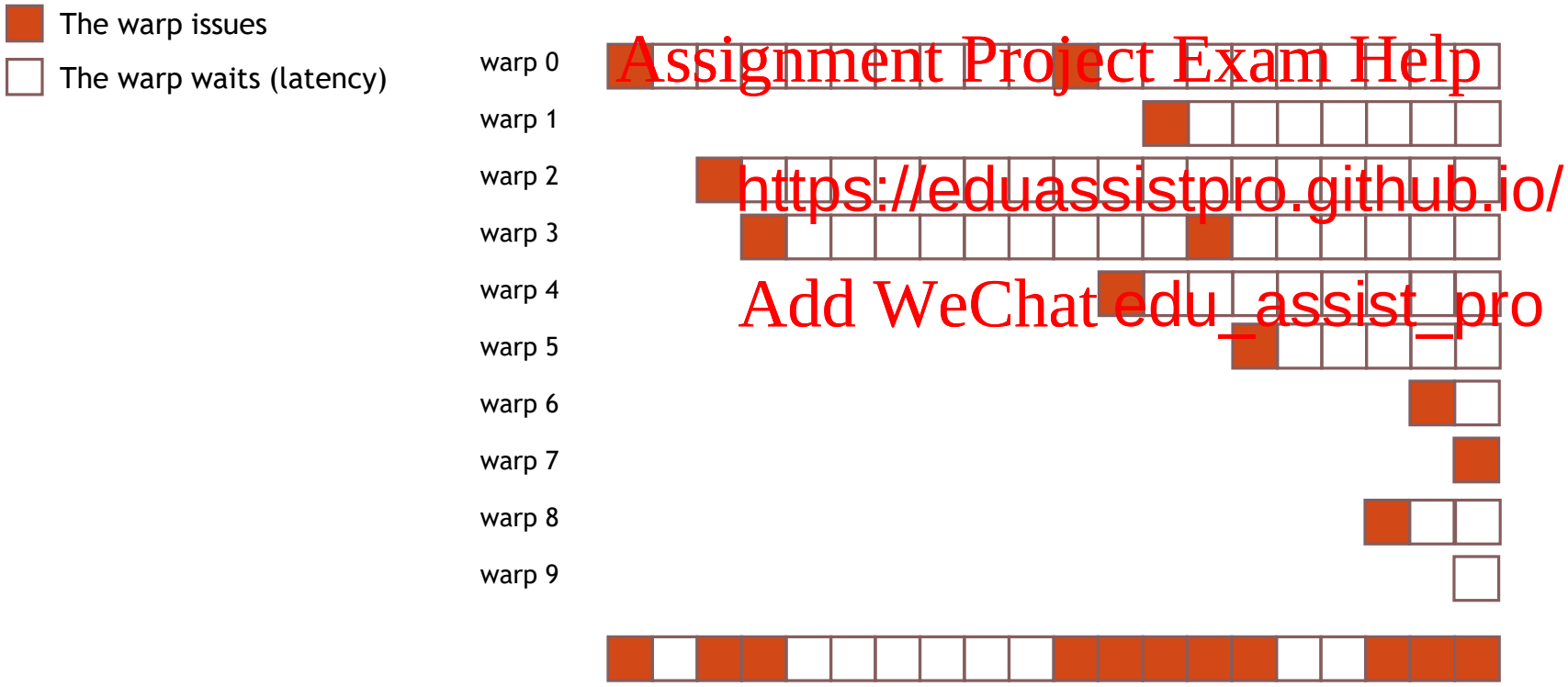
- Occupancy: *"number of active warps over max warps supported"*
- Increasing achieved occupancy can hide latency
  - More warps available for execution = more to hide latency



Occupancy

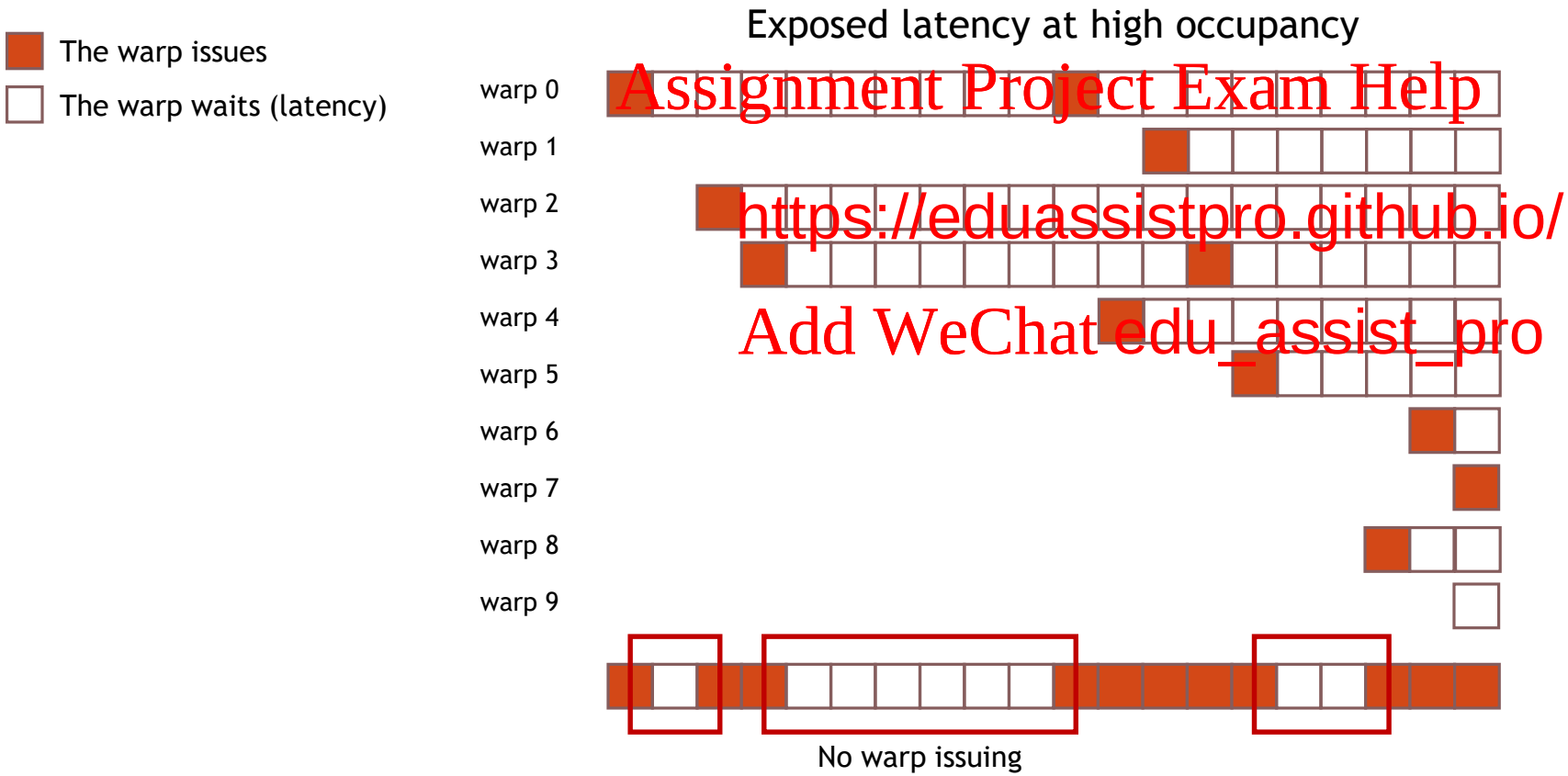
- In our case we are not achieving theoretical occupancy (we have latency)

What is the problem here?



Occupancy

- In our case we have good occupancy but still high latency
  - Schedulers cant find eligible warps at every cycle



Warps are waiting for memory (transactions)

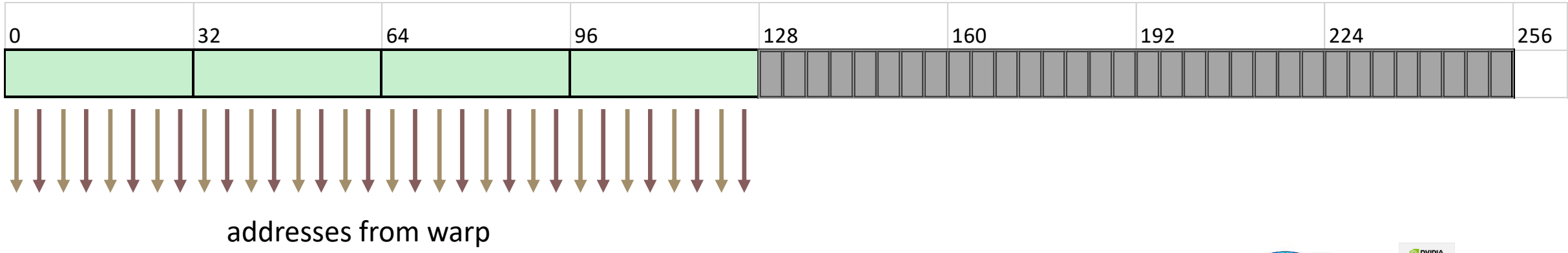
Transaction per access = 5:1

We are using only 20% of the effective bandwidth

Transactions per access?

- Think back to Lecture 11
  - To get 100% efficiency **our threads need to access consecutive 4 byte values**
  - 32 Threads in warp accessing 4B each
    - 128B total via 4 L2 cache lines


```
__global__ void copy(float *
    int xid = blockIdx.x * b
    odata[xid] = idata[xid];
}
```

Profiler is telling that we could use only 1 transaction but are using 4/5 (only 1 transaction required for each thread in warp to read a single byte char)

```
n[j+3][i+3] = in_img(x+i, y+j, w, h) ? (int) src[(y+j)*w + (x+i)] : 0;
```

Memory is indexed based on `x==threadIdx.x`: Suggests access is coalesced.
Cause not clear…..

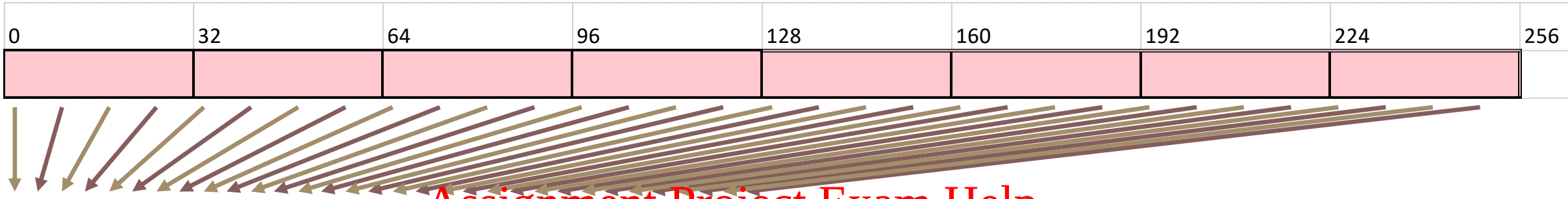
Analysis

- The limiting factor of our code is L2 Throughput
  - There is nothing wrong with having high throughput
  - Except: There is not enough compute to hide this
  - We cant increase occupancy any further to hide this

- Solution: We need to [illegible]es to get data to the device to do compute on it. Either b[illegible]

  - Moving data closer to the SMPs
  - **Making our L2 reads/writes more efficient**
    - Currently ~4-5 Transactions/Access
    - Our L2 cache lines are being used ineffectively

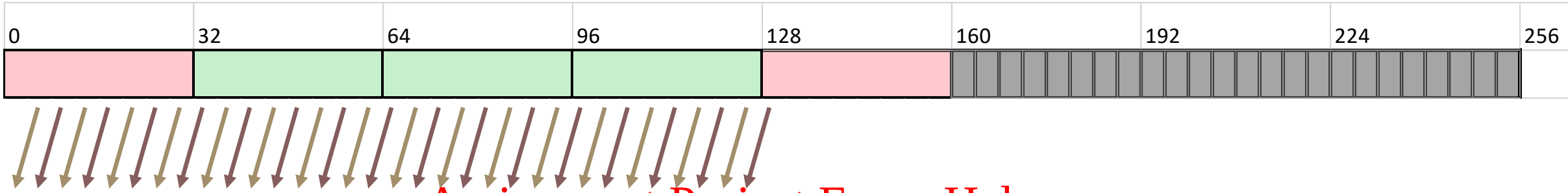
Causes of Transaction per access: Striding?




```
__global__ void cop                                    ta)
    int xid = (bloc                                    xIdx.x)*2;
    odata[xid] = id
}
```


- Lecture 11 example
  - Strides (like above) cause poor transactions per access
  - In the above case 8 transactions where we could have used 4

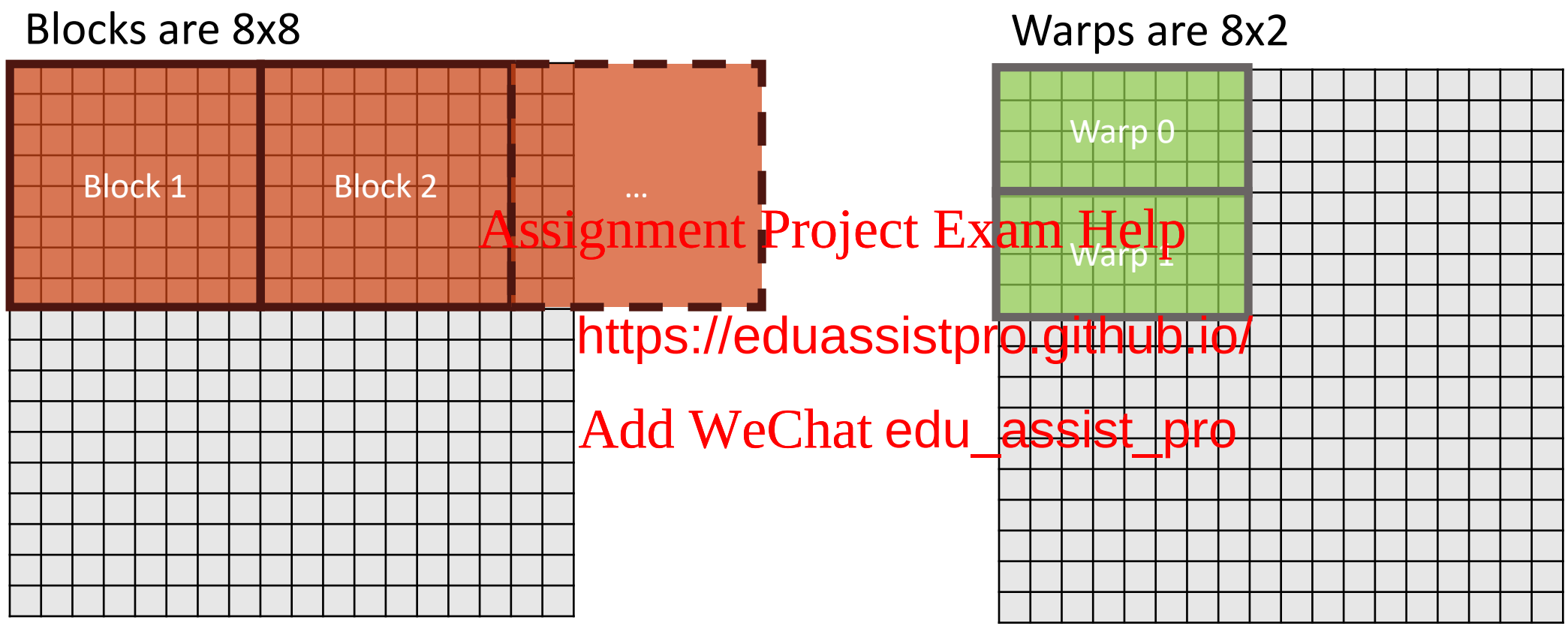
Causes of Transaction per access: Offset?




```
__global__ void copy(float *o
    int xid = blockIdx.x * blockDim.x + threa
    odata[xid] = idata[xid];
}
```

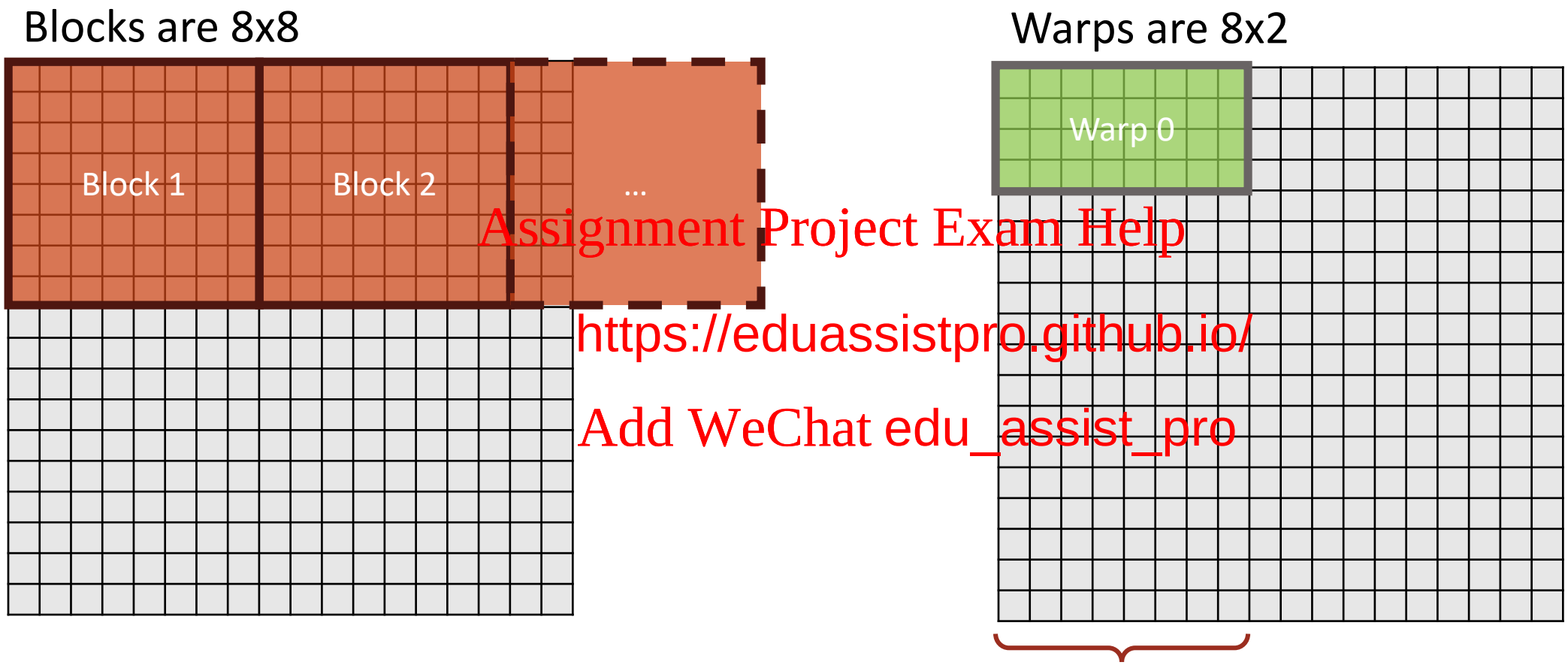

- Lecture 11 Example:
  - If memory accesses are offset then parts of the cache line will be unused (shown in red) e.g.
  - Use thread blocks sizes of multiples of 32!

What is our current data layout?



Why might this be a problem

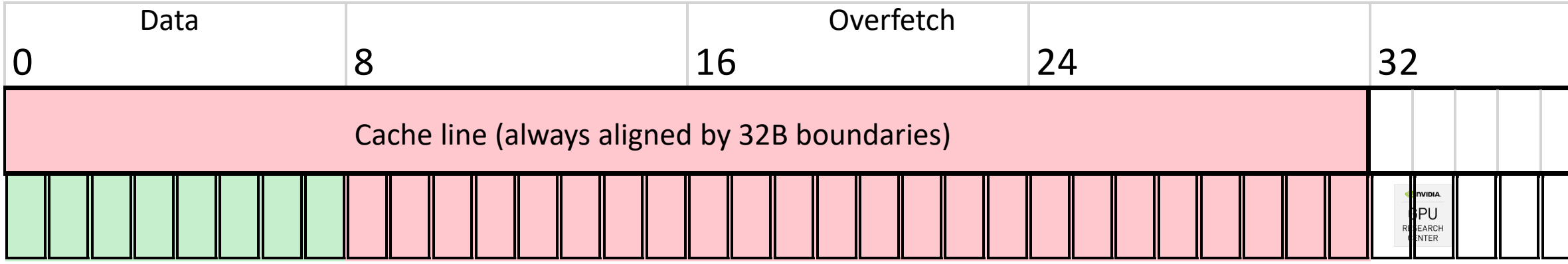
What is our current data layout?



`threadIdx.x` not consecutive within the warp

Overfetch from L2 Cache

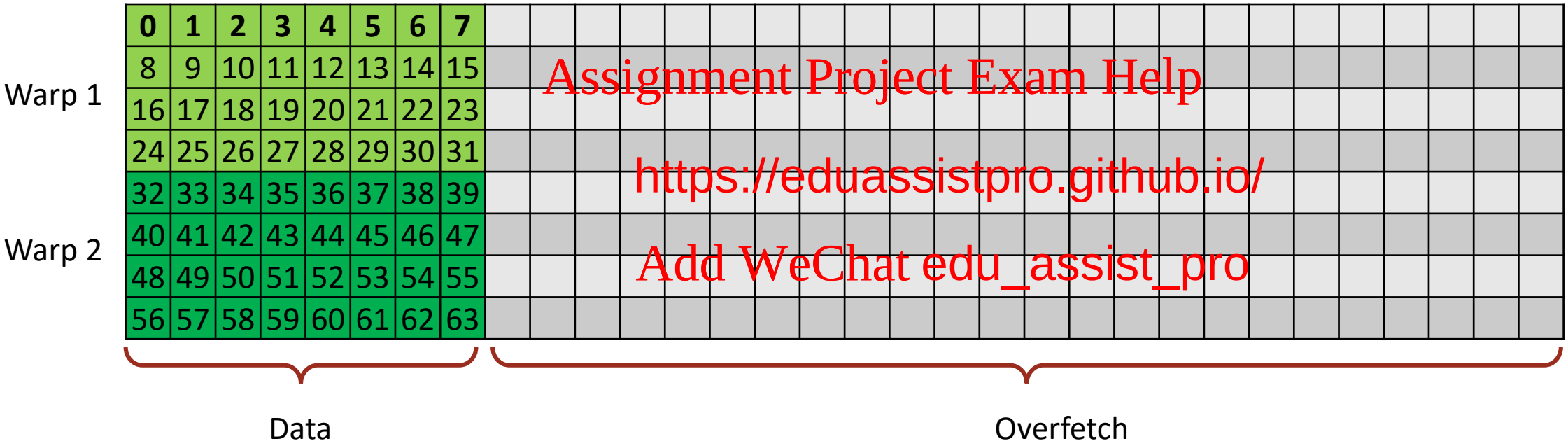
```
Line 245:                 src[(y+j)*w + (x+i)]
Line 245 for i=0, j=0:    src[x]                    //threads 0-7 only
```



Overfetch with L1 Caching

Line 245: **src[(y+j)*w + (x+i)]**

Line 245 for i=0, j=0: **src[y*w + x]**

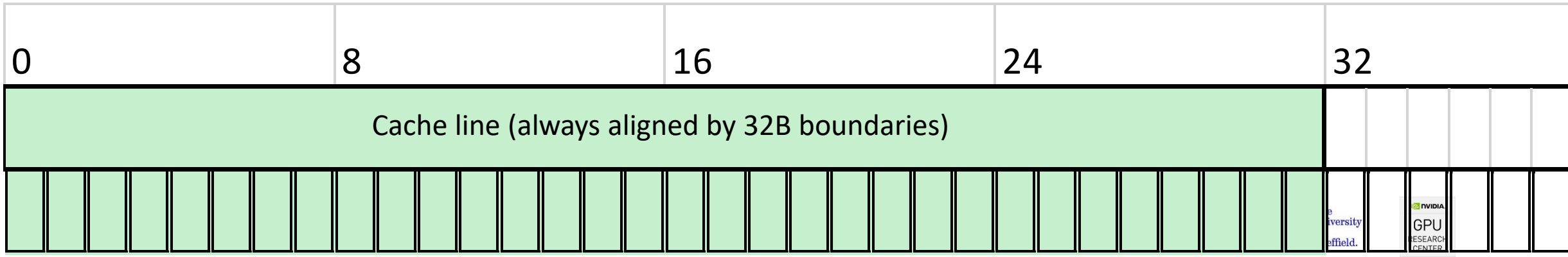


Any Ideas for improving this?

Optimisation: Improved Memory layout

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

- Minimum block width should be 32 (a requires only 1 byte)
- Use Layout of 32x2



Deploy: Improved Memory layout

Kernel	Time (ms)	Speedup	Rel. Speedup
Gaussian_filter (Step 0)			-
Gaussian_filter (Step 1a)			**5.49x**

Break

- What do we expect the analysis to look like next?
- Any ideas for what else may be required?

Half time summary

- The guided profiler will help us optimise the right thing
- Hotspot tells us the most appropriate place to optimise
- Performance Limiter tells us what to focus on to improve
- Code may be Memory Bound
- Improvements so far
  - Changed the access pattern (by changing block size)
  - Reduced memory dependencies?

- Profiling Introduction
- The Problem
- Visual Profiler Guided Analysis
  - Iteration 1
  - Iteration 2

Identify the hotspot

- Examine GPU Usage in Visual Profiler
- Examine Individual Kernels
  - Gaussian filter kernel still the highest rank

Kernel Optimization Priorities

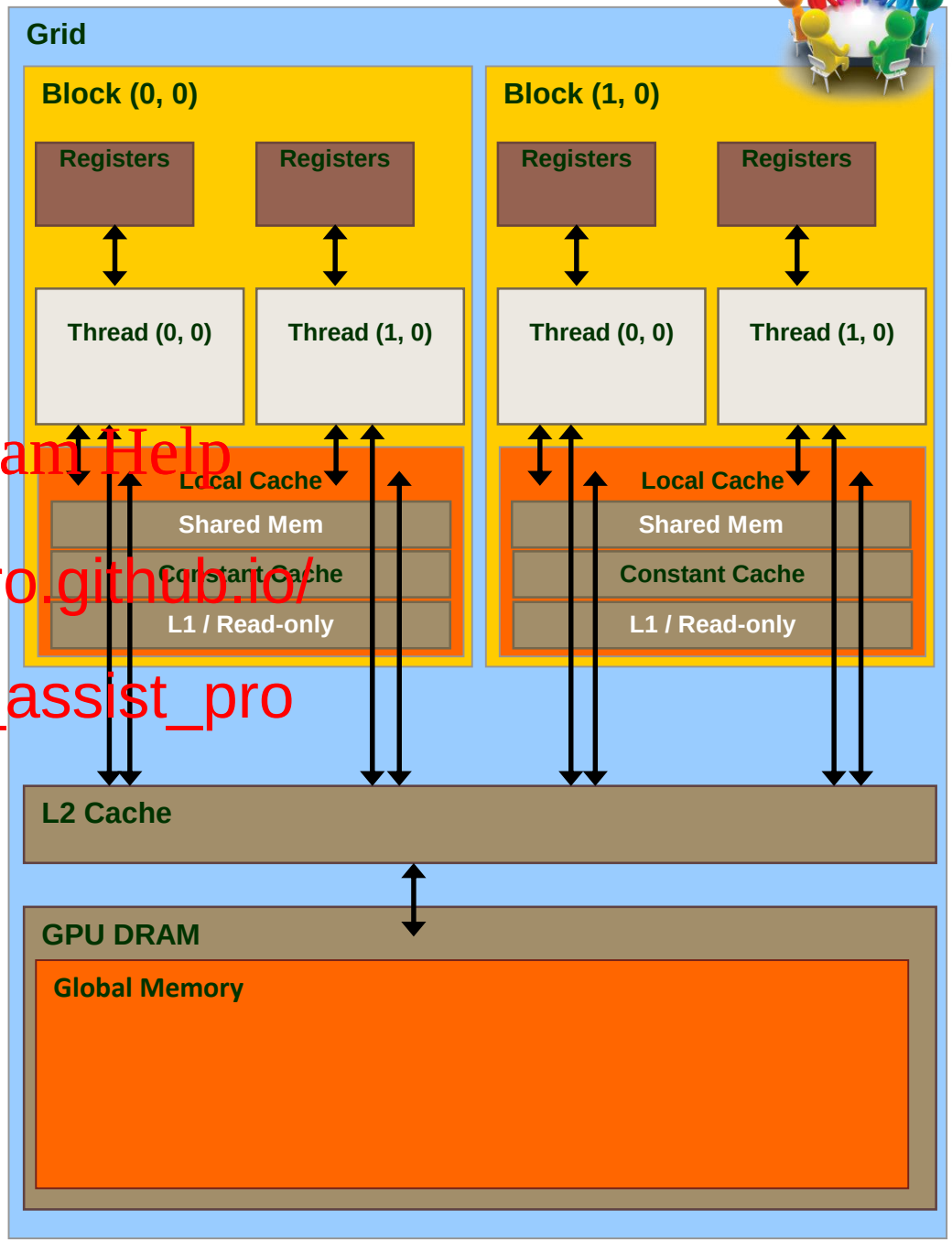
The following kernels are ordered by optimization importance based on execution time and achieved occupancy. Optimization of higher ranked kernels (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[1 kernel instances] gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)
29	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
14	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

Performance Limiter

Tex Instruction Units?

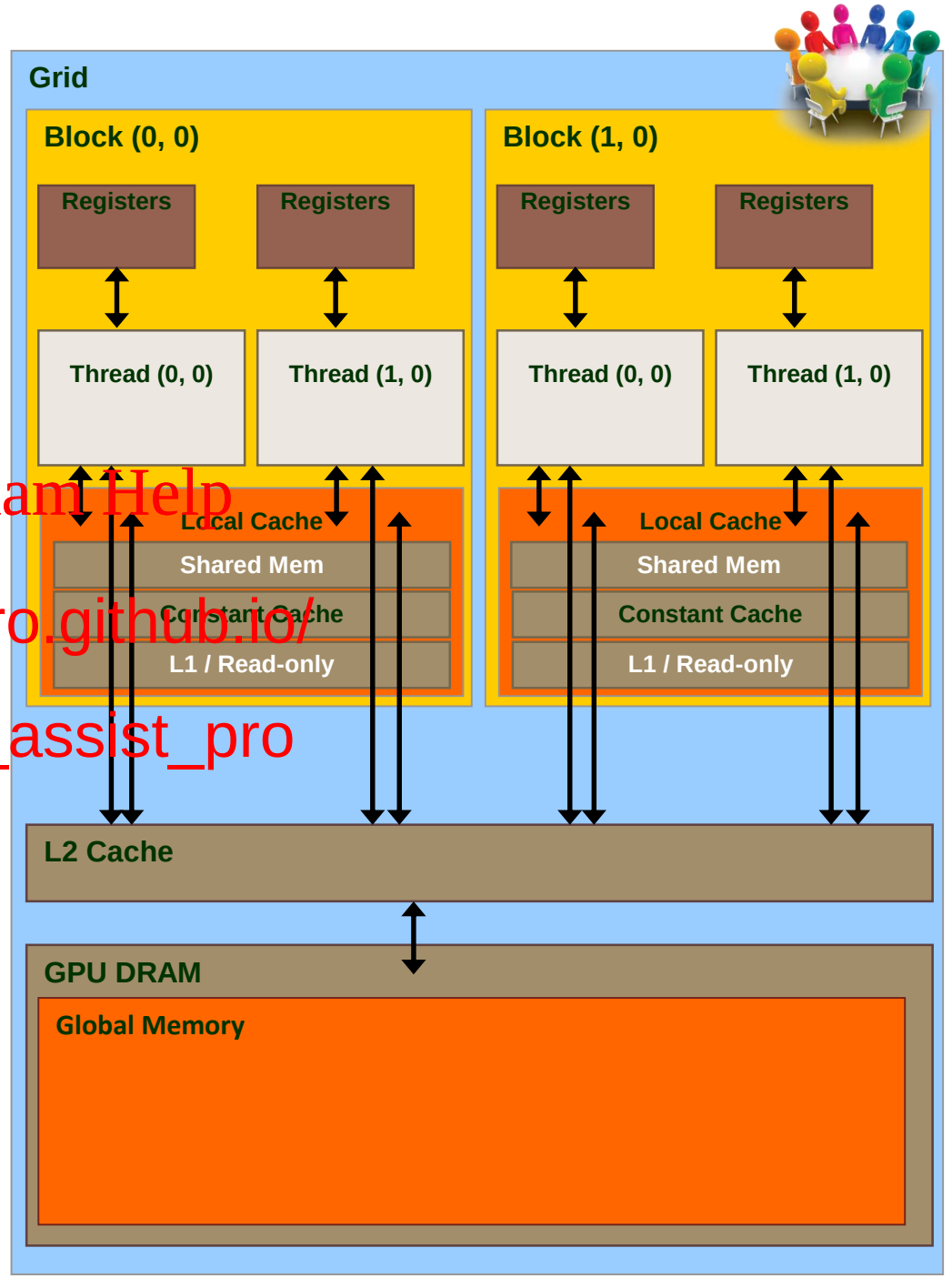
- What are texture instruction units and why might our code be using them?



Tex Instruction Units?

- What are texture instruction units and why might our code be using them?
  - Hint:

```
void gaussian_filter_7x7_v1(int w,
                            int h,
                            const
                            uchar
```

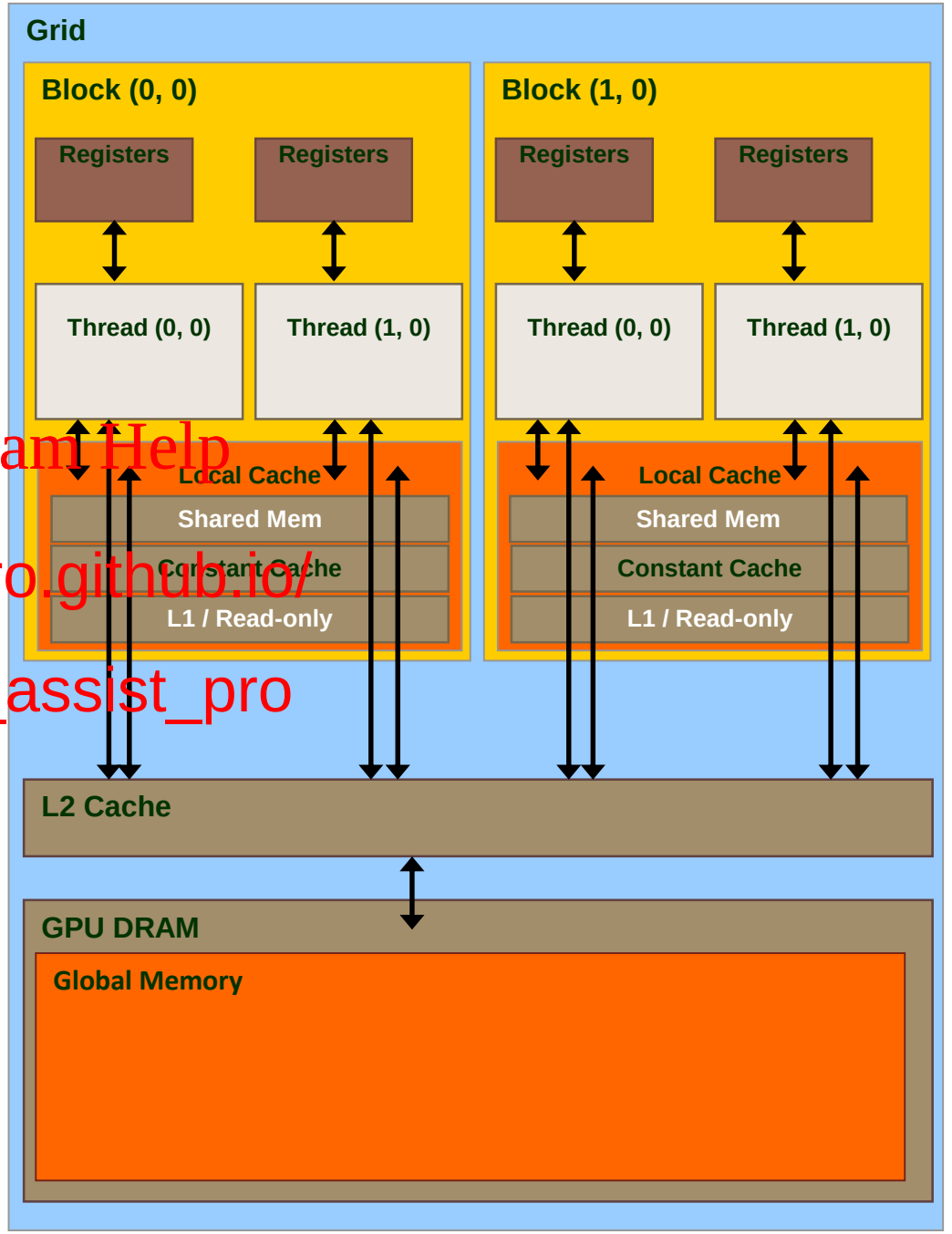



Tex Instruction Units?

- What are texture instruction units and why might out code be using them?

```
void gaussian_filter_7x7_v1(int w,
                            int h,
                            const
                            uchar
```


- Compiler is reading `src` as read-only through Unified L1/Read-Only (texture cache)



Guided Bandwidth Analysis

- We are doing lots of reading/writing through unified cache

Guided Bandwidth Analysis

- Still parts of the code reporting 2 transactions per access?

Transaction per request

```
Line 245:                 src[(y+j)*w + (x+i)]
Line 245 for i=1, j=0:    src[x+1]
```

What is wrong with this access pattern?

Transaction per request

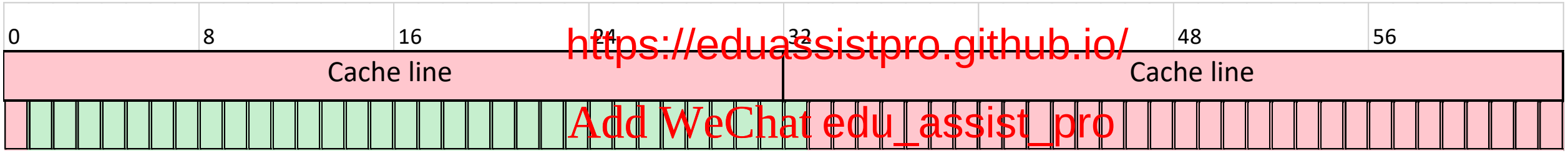
```
Line 245:                 src[(y+j)*w + (x+i)]
Line 245 for i=1, j=0:    src[x+1]
```

What is wrong with this access pattern?

Hint: Cache *oundaries*

Transaction per request

```
Line 245:                 src[(y+j)*w + (x+i)]
Line 245 for i=1, j=0:    src[x+1]
```

We have an offset access pattern

Guided Compute Analysis

- The guided analysis suggests that lots of our compute cycles are spent issuing texture load/stores

Guided Latency Analysis: Occupancy

Guided Latency Analysis: Occupancy

- Register usage is very high
- Occupancy currently limited by register usage

- Increasing occupancy might not help us however as we are dominated
  - More work per SMP w exture load stores!
  - We can confirm this by looking at the unguided analysis: Kernel Latency

PC Sampling

Execution/Memory Dependency

- Rank these are best to worst
- Which have instruction and memory dependencies?


```
int a = b + c;

int d = a + e;

//b, c and e are local ints
```



```
int d = a + e;

//b is global memory
//I and e are local in
```



```
int a = b + c;

int d = e + f;

//b, c, e and f are local ints
```

Instruction/Memory Dependency

- Rank these are best to worst
- Which have instruction and memory dependencies?

```
int a = b + c;

int d = a + e;

//b, c and e are local ints
```

- Instruction Dependency
- Second add must wait for first

```
int d = a + e;

//b is global memory
//i and e are local in
```


- Memory Dependency
- Second add must wait for memory request

```
int a = b + c;

int d = e + f;

//b, c, e and f are local ints
```

- No dependencies
- Independent Adds

Analysis

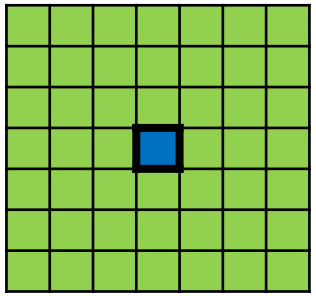
- Our compute engine is dominated by load/store instructions for the texture cache
  - Our texture bandwidth is good BUT
- Our warps are stalling as instructions are waiting to issue texture fetch instructions
- We still have poorly a within our inner loops
- Solution: Reduce dependencies on texture loads
  - Move data closer to the SMP
  - Only read from global memory with nicely aligned cache lines
  - **How?**

Analysis

- Our compute engine is dominated by load/store instructions for the texture cache
  - Our texture bandwidth is good BUT
- Our warps are stalling as instructions are waiting to issue texture fetch instructions
- We still have poorly a within our inner loops
- Solution: Reduce dependencies on texture loads
  - Move data closer to the SMP
  - Only read from global memory with nicely aligned cache lines
  - **Shared Memory**

Shared Memory

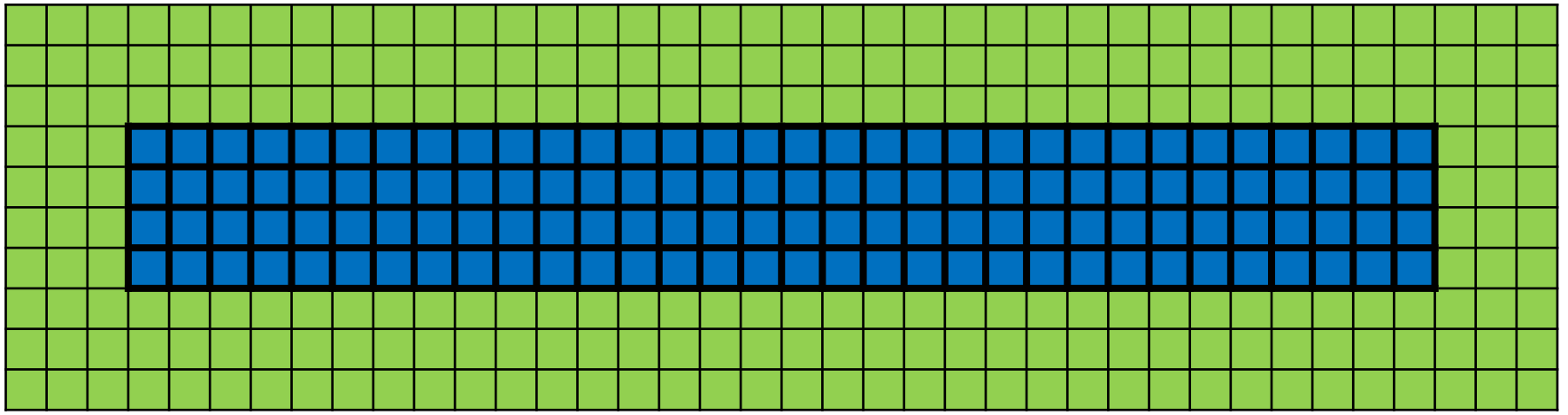
Single thread uses 7x7= 42 values



Use shared memory to store all pixels for the block

What important factor should we be considering?

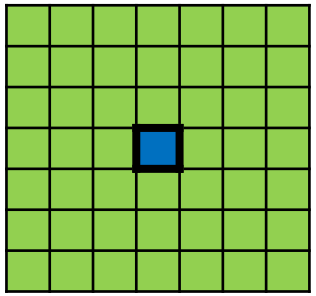
Single block (32x4) uses 38x10 = 680 values



Also increased Block size

Shared Memory

Single thread uses 7x7= 42 values

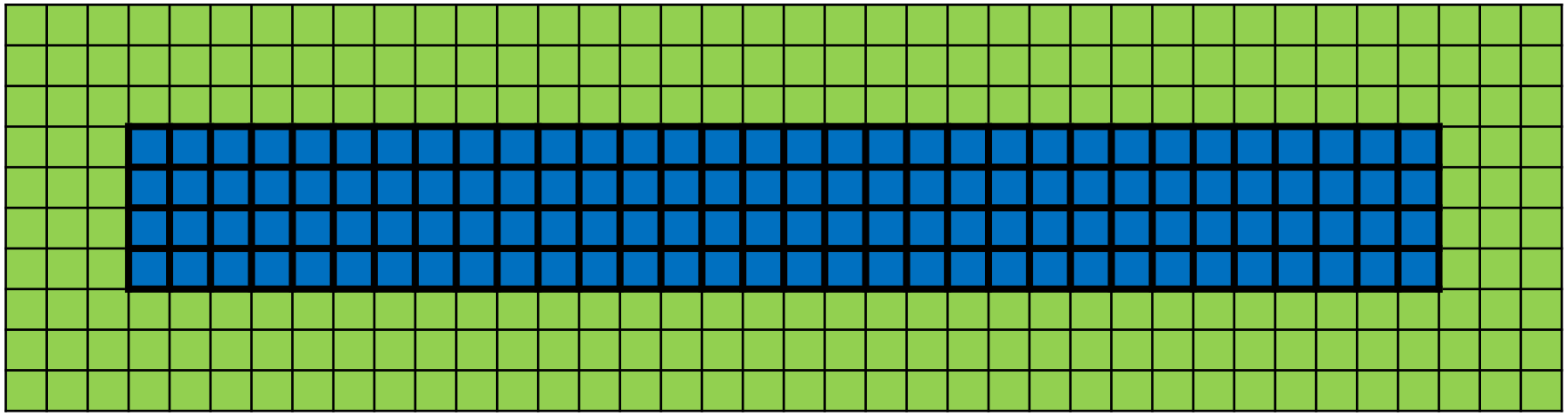


Use shared memory to store all pixels for the block

```
__shared__ unsigned char smem_pixels[10][64]
```


SM bank conflicts

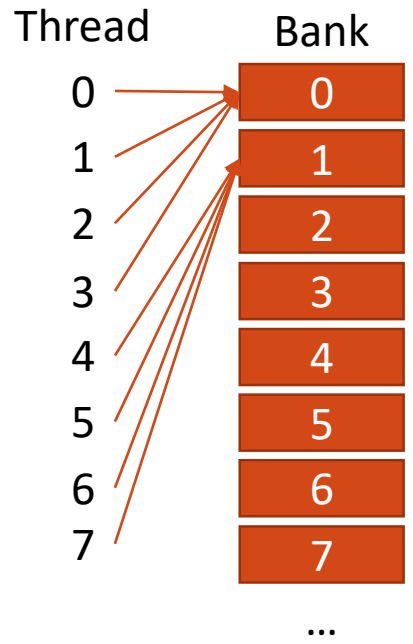
Single block (32x4) uses 38x10 = 680 values



Also increased Block size

BUT WAIT!!!!!!!!!!!!!!!!

- Wouldn't aligned char access have 4 way bank conflicts?
  - NOT for Compute Mode 2.0+…



"A shared memory request for a warp does not generate a bank conflict between two threads that access any address with *hough the two addresses fa* *e, for read accesses, the word is* ***broadcas*** *ting threads (multiple words can be* *single transaction) …"*

I.e. A Stride of less than 1 (4B word) can be read conflict free if threads access aligned data

Improvement

- Significant

Kernel	Time (ms)	Speedup	Rel. Speedup
Gaussian_filter (Step 0)	5.49	1.00x	-
Gaussian_filter (Step 1a)			5.49x
Gaussian_filter (Step 40)	**0.49**		**2.04x**

- Profiling Introduction
- The Problem
- Visual Profiler Guided Analysis
  - Iteration 1
  - Iteration 2
  - Iteration 3

Identify the hotspot

- Examine GPU Usage in Visual Profiler
- Examine Individual Kernels
  - Gaussian filter kernel still the highest rank
    - Getting much closer though

Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved occupancy. Optimization of higher ranked kernels (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[1 kernel instances] gaussian_filter_7x7_v2(int, int, unsigned char const *, unsigned char*)
60	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
29	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

Performance Limiter

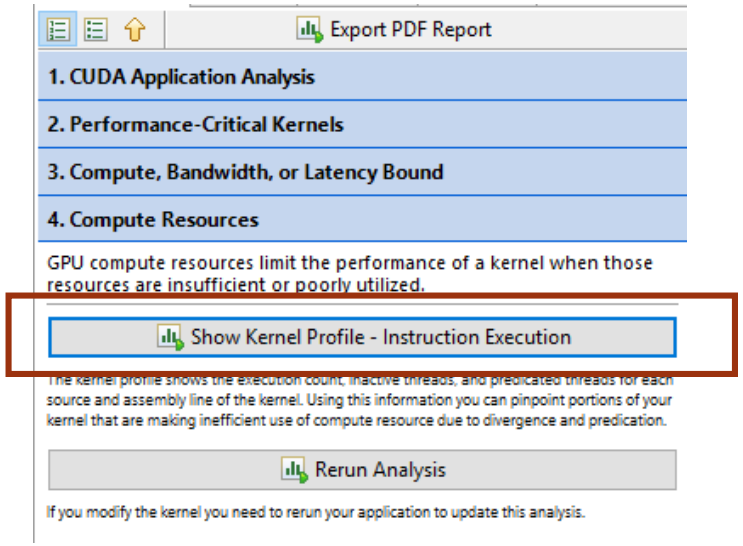
- Actually very close to magical 60% of compute
- Lets examine
  - 1) The compute analysis
  - 2) The latency analysis

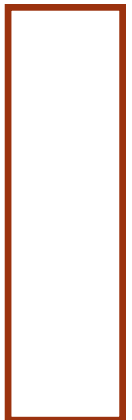
Guided Bandwidth Analysis

Compute Analysis

- We are simply doing lots of compute
- Additional floating point operations graph shows no activity i.e. all of our instructions are Integer

What are all of these integer instructions?





The University Of Sheffield.

NVIDIA GPU RESEARCH CENTER

Compute Analysis by Line

- Selecting the CUDA function from compute analysis results allows a line by line breakdown
  - This will switch to unguided analysis

Also PTX instruction breakdown provided

Guided Latency Analysis

Would changing the block size, register usage or amount of shared memory per block improve occupancy?

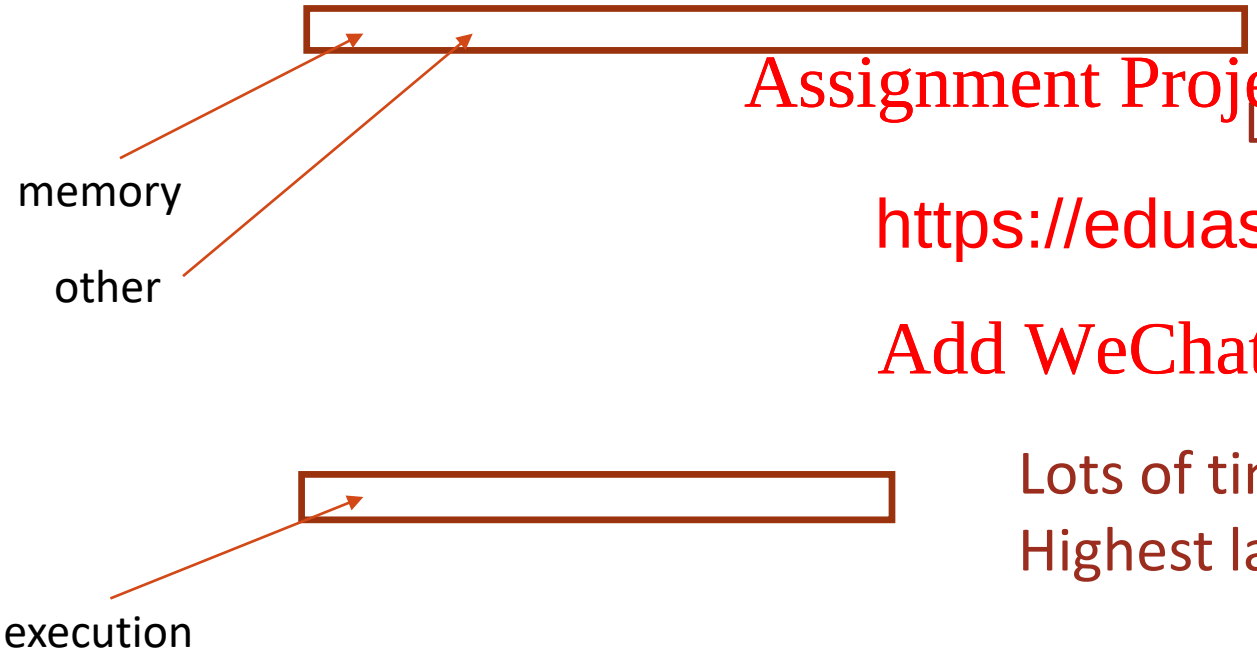
Guided Latency Analysis

Line by Line Breakdown

Latency Overview: Other 32.25%

- Stall reason other generally means that there is no obvious action to improve performance
- Other stall reasons may indicate either;
  1. Execution unit is busy
     - Solution: Potentially r teger operations if possible
  2. Register bank conflic an sometimes be made worst by heavy use of vector data types
     - Solution: None
  3. Too few warps per scheduler
     - Solution: Increase occupancy, decrease latency

Guided Latency Analysis: Line by Line



Lots of time spent loading into shared memory
latency is on memory

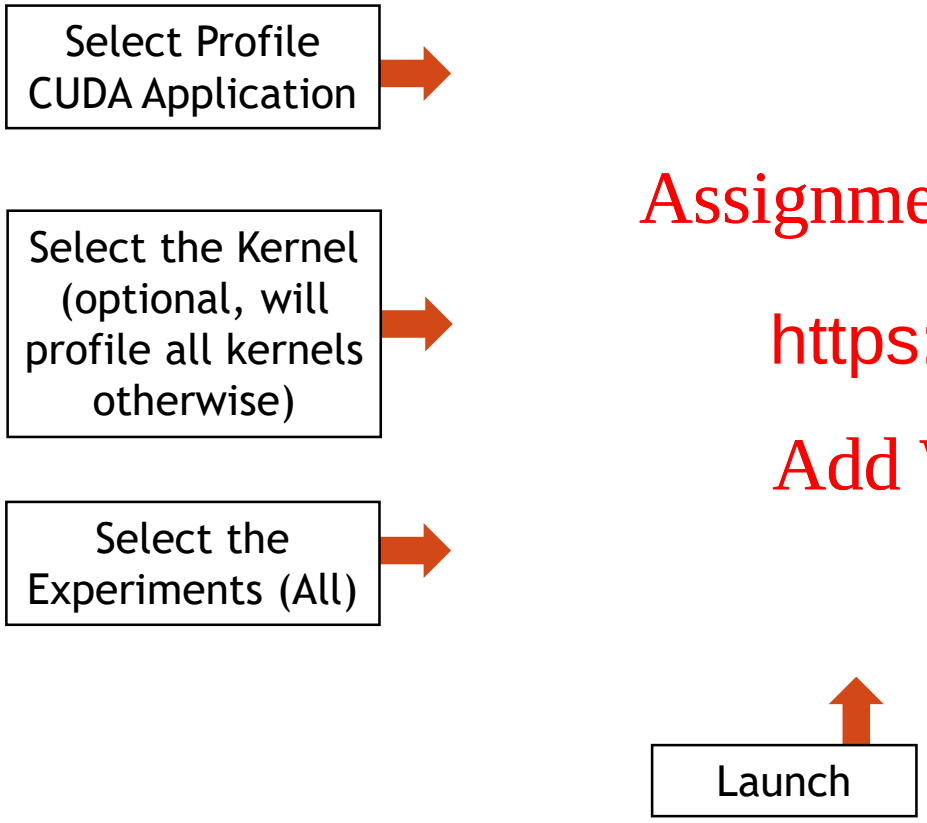
Lots of time spent computing convolution
Highest latency is on execution dependency

1st Analysis

- We have a reasonably well balanced use of the from Compute and Memory pipes.
- There is some latency in loading data to and from shared memory
- Our compute cycles are dominated by integer operations
  - What operations are t
  - We can either examin ctions (from Compute or Latency Analysis) or run additional ana ht within Visual Studio
    - More detailed analysis
    - Not guided like the visual profiler

Start profiling

Kernel Analysis



CUDA Launches View

Performance Indicators

Achieved IOPS

- No surprises…

```
int p = 0;
 for( int j = 0 ; j < 7 ; ++j )
   for( int i = 0 ; i < 7 ; ++i )
     p += gaussian_filter[j][i] * n[j][i];
```

Pipe Utilisation

- More detailed confirmation
  - Integer operations dominate

Issue Efficiency

- This is good
- We have no divergent code

2nd Analysis

- We have a reasonably well balanced use of the from Compute and Memory pipes.
- **There is some latency in loading data to shared memory and on executions to read it back**
- **Our compute cycles a ger operations**

There is some latency in loading data to shared memory and on executions to read it back

- Consider a simplified problem
- Each thread needs to load an r, g, b, a value into shared memory
  - Which has fewer shared memory load instructions?


```
__shared__ char sm[TPB*4];

char r,g,b,a;

r = sm[threadidx.x];
g = sm[threadidx.x+1];
b = sm[threadidx.x+2];
a = sm[threadidx.x+3];
```

```
__sh                    PB];
char
char4 rgba;
rgba = sm[threadidx.x];
r = rgba.r;
g = rgba.g;
b = rgba.b;
a = rgba.a;
```

There is some latency in loading data to shared memory and on executions to read it back

- Consider a simplified problem
- Each thread needs to load an r, g, b, a value into shared memory
  - Which has fewer shared memory load instructions?


```
__shared__ char sm[TPB*4];

char r,g,b,a;

r = sm[threadidx.x];
g = sm[threadidx.x+1];
b = sm[threadidx.x+2];
a = sm[threadidx.x+3];
```

```
__sh                    PB];
char
char4 rgba;
rgba = sm[threadidx.x];
r = rgba.r;
g = rgba.g;
b = rgba.b;
a = rgba.a;
```

Our compute cycles are dominated by Integer operations

```
int p = 0;
 for( int j = 0 ; j < 7 ; ++j )
   for( int i = 0 ; i < 7 ; ++i )
     p += gaussian_filter[j][i] * n[j][i];
```


- Which of the followin


```
int a, b, c;
a = sm_a[i]; b = sm_b[i];

c += a * b;
```

```
int a, b, c;
a = sm_a[i]; b = sm_b[i];

c += a * b;
```

Our compute cycles are dominated by Integer operations

```
int p = 0;
 for( int j = 0 ; j < 7 ; ++j )
   for( int i = 0 ; i < 7 ; ++i )
     p += gaussian_filter[j][i] * n[j][i];
```


- Which of the followin


```
int a, b, c;
a = sm_a[i]; b = sm_b[i];

c += a * b;
```

```
int a, b, c;
a = sm_a[i]; b = sm_b[i];

c += a * b;
```

Integer multiply add is 16 cycles

Float combined multiply add is 4 cycles

Analysis

- We have a reasonably well balanced use of the from Compute and Memory pipes.
- There is some latency in loading data to shared memory and on executions to read it back
  - **Solution 1**: Reduce SM Load Stores dependencies by using wider requests. i.e. 4B values rather th
  - I.e. Store shared mem
- Our compute cycles are dominated by Integer operations
  - Almost all MAD operations
  - **Solution**: Change slower Integer MAD instructions to faster floating point FMAD instructions
  - I.e. Use floating point multiply and cast result to `uchar` at end

Improvement

- Significant

Kernel	Time (ms)	Speedup	Rel. Speedup
Gaussian_filter (Step 0)	5.49	1.00x	-
Gaussian_filter (Step 1a)			5.49x
Gaussian_filter (Step 40)	0.49		2.04x
Gaussian_filter (Step 5a)	**0.28**		**1.75x**

- Profiling Introduction
- The Problem
- Visual Profiler Guided Analysis
  - Iteration 1
  - Iteration 2
  - Iteration 3
  - Iteration 4

Identify the hotspot

- Examine GPU Usage in Visual Profiler
- What should be our next step?

- Lets look at our Gaussian kernel anyway…

Identify the hotspot

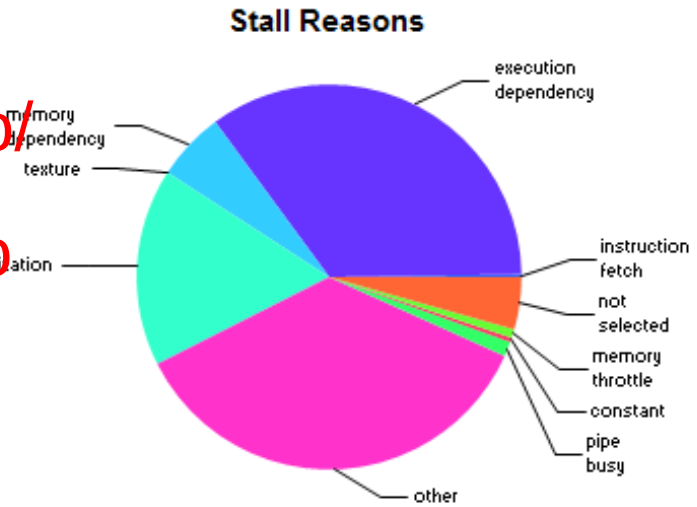
- Examine GPU Usage in Visual Profiler
- Examine Individual Kernels
  - Gaussian filter kernel no longer highest rank!
  - We can now optimise the sobel_filter kernel

- Lets look at our Gaussian kernel anyway…

Performance Limiter

- Looking good

VS NSight IOPS/ FLOPS Metrics



Analysis

- Our algorithm is making good use of compute and memory
- Further improvement will be difficult (but not impossible)

- **Solution**: Optimise a
  - sobel_filter_kernel to
- **Solution**: Improve Gaussian kernel g the technique (parallelise differently)
  - Separable Filter: Compute horizontal and vertical convolution separately then approximate by binominal coefficients
  - Ensure we apply the same optimisations to separable filter version

Improvement

Kernel	Time (ms)	Speedup	Rel. Speedup
Gaussian_filter (Step 0)	5.49	1.00x	-
Gaussian_filter (Step 1a)			5.49x
Gaussian_filter (Step 40)	0.49		2.04x
Gaussian_filter (Step 5a)	0.28		1.75x
Gaussian_filter (Step 9)	**0.22**	**24.95x**	**1.27x**

- 25x speedup on existing GPU code is pretty good

- Companion Code: https://github.com/chmaruni/nsight-gtc

Summary

- Profiling with the Visual Profiler will give you guided analysis of how to improve your performance
  - Show you how to spot key metrics
- **We are trying to achieve good overall utilisation of the hardware (compute and memory engines)**
  - **Through an appreciat ute bounds**
- Follow the APOD cycle
  - Assess: What is the limiting factor, an ofile
  - Parallelise and improve (apply the knowledge you have learnt over the course)
  - Optimise
  - Deploy and Test
- If in doubt use the lab classes to seek guidence!