Occupancy Considerations

Dr. Alptekin Temizel

Assignment 2- Automatic Contrast Enhancement

- Assignment will be uploaded to ODTUClass today
- Deadline 14th April

Assignment 2- Automatic Contrast Enhancement

$$pDst(i,j) = \frac{pSrc(i,j) - nMin}{nMax - nMin} x\ 255$$

Assignment 2

- As you can see the operation could be divided into 3 parts:
  - Finding *nMin* and *nMax*
  - Subtracting *nMin* from the pixels of the source image *pSrc*
  - Scaling the pixels of source image *pSrc* by a factor of

$$\frac{255}{nMax - nMin}$$

Assignment 2

- In the supplied code you'll see that these operations have been done using the NPP library. You can run this code and observe the results.
- Now you'll do the same thing on
  - CPU
  - On GPU using CUDA
- You'll then benchmark your results and compare.

Notes

- Npp8u type is basically 8-bit unsigned chars used in NPP library (typedef unsigned char Npp8u), you can use this type in your code.

- Use QueryPerformanceCounter function in windows for high precision timing for CPU, see the following link for an example:

http://stackoverflow.com/questions/1739259/how-to-use-queryperformancecounter

- $\frac{255}{nMax-nMin}$: You can do this by integer arithmetic, please have a look at GTC 2013 Lab NPP.pptx

- You can use the calculation that is already in the code. You need to multiply by *nConstant* and then divide by *nScaleFactor*-1.

Notes- Common Errors

- Be careful about the data types you use, type mismatch, especially between **unsigned** and **signed** types will result in the code to work but generate strange results (like distorded images).

- If there is a **pitch** parameter, be careful about what you set.

Quiz #3

- Quiz #3 will be on 9th April at 9:40
- Please be on time. If you are late, you will not be compensated for late arrival
- The quiz questions will be from the contents of this week's slides (both parts), so please study these slides carefully

Launch Configuration

- Instructions are issued in order
- A thread stalls when one of the operands isn't ready
- Latency is hidden by switching warps (32 threads)
  - Global memory latency: 400-800 cycles
  - Need enough threads to hide latency

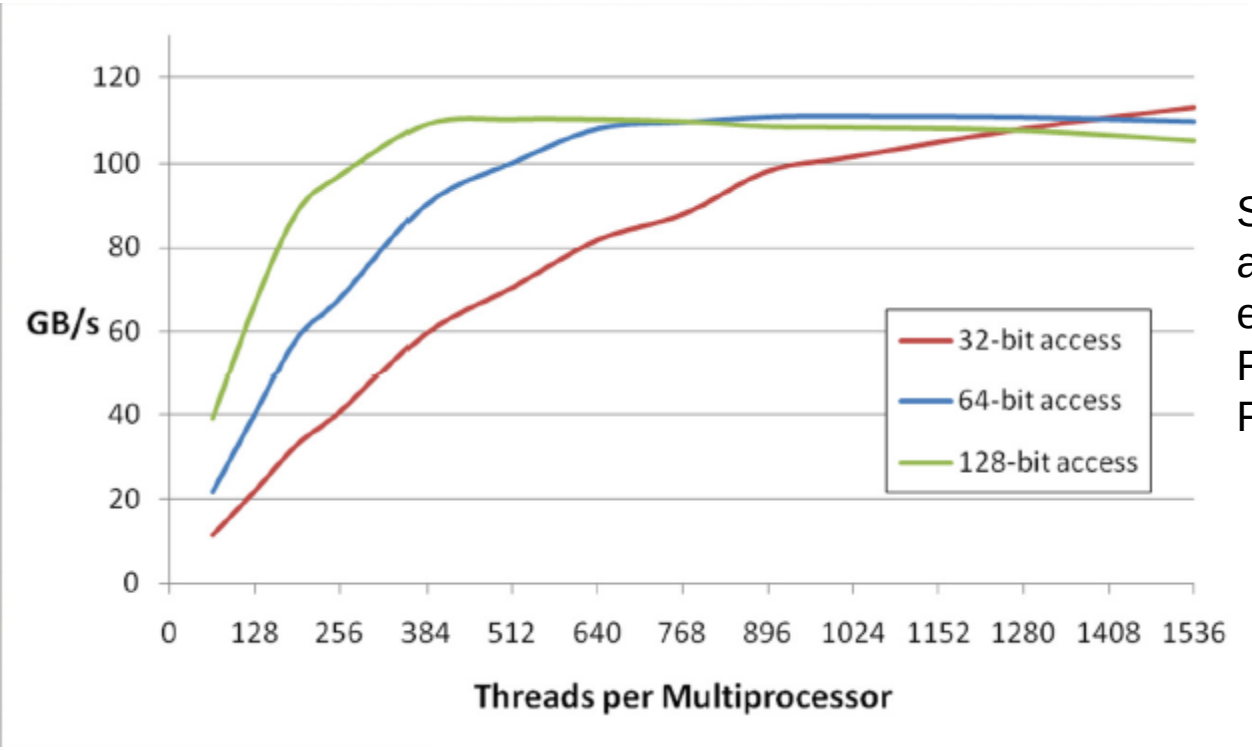
Launch Configuration

How many threads/threadblocks to launch?

- Number of threads needed depends on:
  - access pattern
  - word size
- Need enough memory transactions in flight to saturate the bus
- Increase transactions by having
  - Independent loads and stores from the same thread
  - Loads and stores from different threads (more threads)
  - Larger word sizes (float2 is twice the transactions of float, for example)

Maximizing Memory Throughput

- Increment an array of 64M elements
  - Two accesses per thread (load then store)
  - The two accesses are dependent
- Tesla C2050, ECC on, theoretical bandwidth: ~120 GB/s



Several independent smaller accesses have the same effect as one larger one.
For example:
Four 32-bit ~= one 128-bit

Occupancy

- Occupancy = Active Warps / Maximum Active Warps

- Resources are allocated for the entire block
  - Resources are finite
  - Utilizing too many resources per thread may limit the occupancy (by limiting the number of active warps)

Occupancy

Potential occupancy limiters:

- Register usage
- Shared memory usage
- Block size

Occupancy

Technical specifications	Compute capability (version)						
	1.0	1.1	1.2	1.3	2.x	3.0	3.5
Maximum dimensionality of grid of thread blocks	2				3		
Maximum x-, y-, or z-dimension of a grid of thread blocks	65535					2^{31}-1	
Maximum dimensionality of thread block	3						
Maximum x- or y-dimension of a block	512				1024		
Maximum z-dimension of a block	64						
Maximum number of threads per block	512				1024		
Warp size	32						
Maximum number of resident blocks per multiprocessor	8					16	
Maximum number of resident warps per multiprocessor	24		32		48	64	
Maximum number of resident threads per multiprocessor	768		1024		1536	2048	
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K	64 K	
Maximum number of 32-bit registers per thread	128				63		255
Maximum amount of shared memory per multiprocessor	16 KB				48 KB		
Number of shared memory banks	16				32		
Amount of local memory per thread	16 KB				512 KB		
Constant memory size	64 KB						
Cache working set per multiprocessor for constant memory	8 KB						

Occupancy Limiters: Shared Mem.

- Shared memory usage: compile with --ptxas-options=-v
  - Reports shared memory per block
- Fermi has either 16K or 48K shared memory
- Example 1, 48K shared memory
  - Kernel uses 32 bytes of shared memory per thread
  - 48K/32 = 1536 threads
    - Occupancy = 1
- Example 2, 16K shared memory
  - Kernel uses 32 bytes of shared memory per thread
  - 16K/32 = 512 threads
  - occupancy=.3333
- Don’t use too much shared memory
- Choose L1/Shared config appropriately.

Occupancy Limiters: Registers

- Register usage: compile with --ptxas-options=-v
- Fermi has 32K registers per SM

- Example 1
  - Kernel uses 20 registers per thread (+1 implicit)
  - Active threads = 32K/21 = 1560 threads
    - > 1536 thus an occupancy of 1

Occupancy Limiters: Registers

- Example 2
  - Kernel uses 63 registers per thread (+1 implicit)
  - Active threads = 32K/64 = 512 threads
    - 512/1536 = .3333 occupancy – no of warps reduced to 1/3!

Occupancy Limiters: Registers

- Register spills: Using local memory (which is off chip) for excess registry items.
- Doesn’t always decrease performance, but it will:
  - Increase pressure on the memory bus
  - Increase instruction count

Occupancy Limiters: Registers

- Can control register usage with
  - the nvcc flag: --maxrregcount
  - Using `__launch_bounds__()` directives on individual kernels right in the source code.

- Limiting the number of registers used can be useful to increase occupancy, which may or may not make your kernel run faster.

Occupancy Limiters: Registers

If a kernel grid is already running at least one thread block per multiprocessor in the GPU, and it is bottlenecked by computation and not by global memory accesses, then increasing occupancy may have no effect.

Occupancy Limiters: Block Size

- Each SM can have up to 8 active blocks
- A small block size will limit the total number of threads
- Avoid small block sizes, generally 128-256 threads is sufficient

Block Size	Active Threads	Occupancy
32	256	.1666
64	512	.3333
128	1024	.6666
192	1536	1
256	2048 (1536)	1

What Occupancy Do I Need?

- Depends on your problem…
  - Many find 66% is enough to saturate the bandwidth
- Look at increasing occupancy only if the following are true!
  - The kernel is bandwidth bound
  - The achieved bandwidth is significantly less than peak
- Instruction Level Parallelism (ILP) can have a greater effect than increasing occupancy
- Vasily Volkov's GTC2010 talk "Better Performance at Lower Occupancy":

http://www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf

Occupancy Calculator

- A useful profiling tool which can help you investigate occupancy, throughput, and bandwidth.

- Measures actual occupancy and thus may detect problems that shouldn't appear in theory

CUDA_Occupancy_Calculator.xls