CS 295: Modern Systems
GPU Computing Introduction

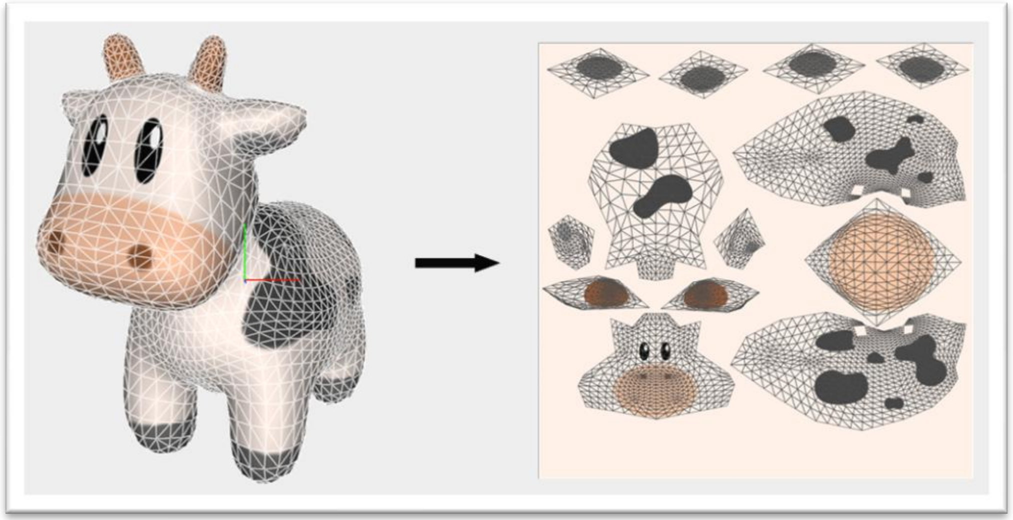
Sang-Woo Jun

Spring 2019

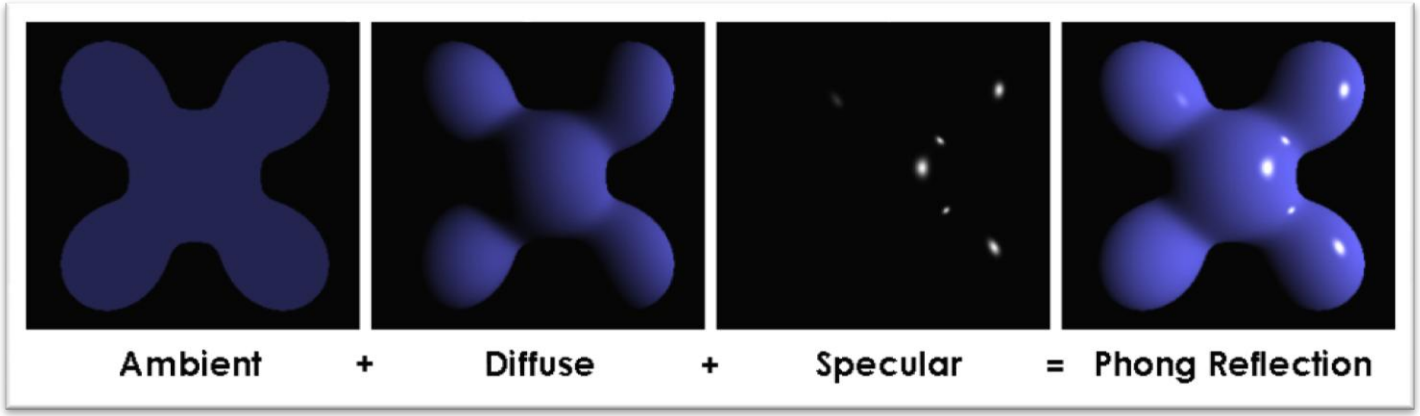
UCI

Graphic Processing – Some History

- ❑ 1990s: Real-time 3D rendering for video games were becoming common
  - ○ Doom, Quake, Descent, … (Nostalgia!)
- ❑ 3D graphics processing is immensely computation-intensive



Texture mapping



Shading

Warren Moore, “Textures and Samplers in Metal,” Metal by Example, 2014

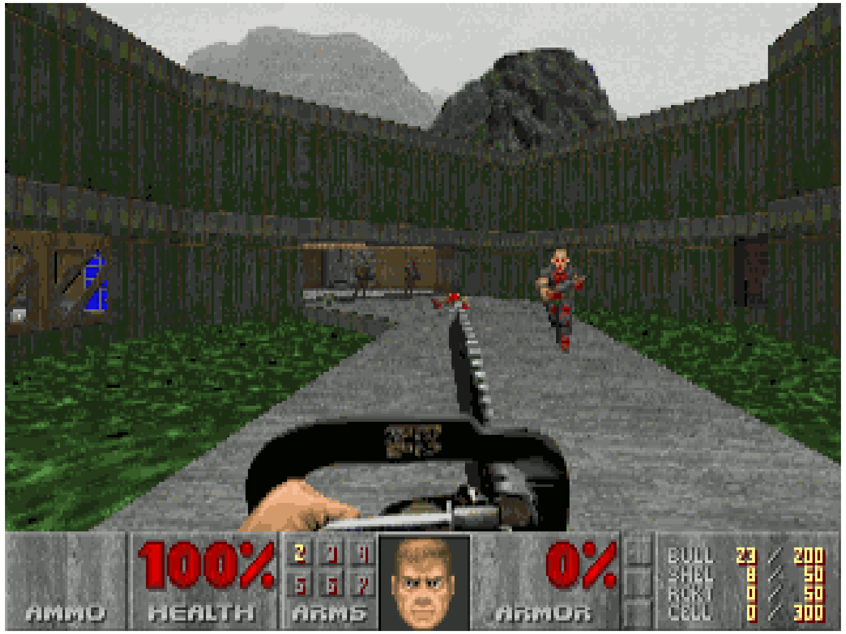
Gray Olsen, “CSE 470 Assignment 3 Part 2 - Gourad/Phong Shading,” grayolsen.com, 2018

Graphic Processing – Some History

❑ Before 3D accelerators (GPUs) were common

❑ CPUs had to do all graphics computation, while maintaining framerate!

- Many tricks were played



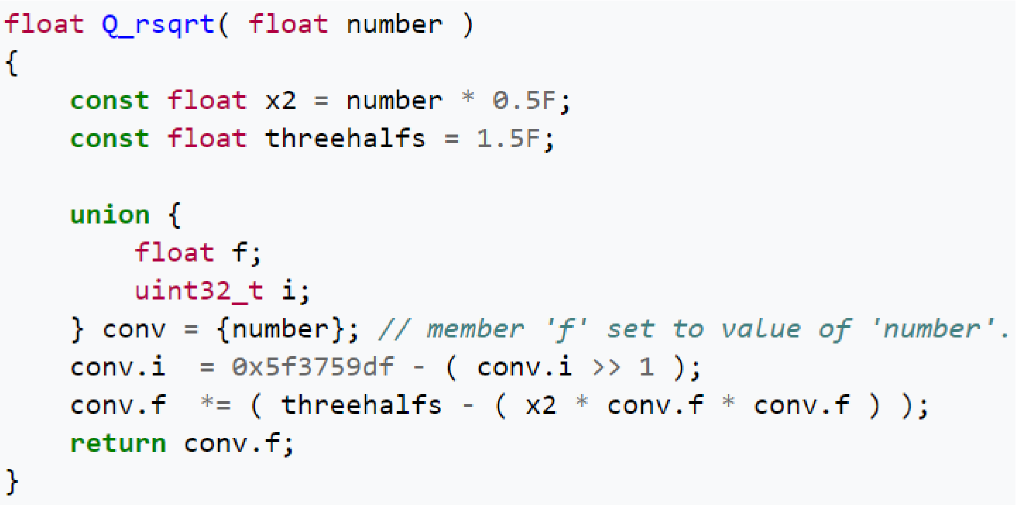
Doom (1993) : “Affine texture mapping”

- Linearly maps textures to screen location, disregarding depth
- Doom levels did not have slanted walls or ramps, to hide this

Graphic Processing – Some History

- Before 3D accelerators (GPUs) were common
- CPUs had to do all graphics computation, while maintaining framerate!
  - Many tricks were played

Quake III arena (1999) : “Fast inverse square root” magic!

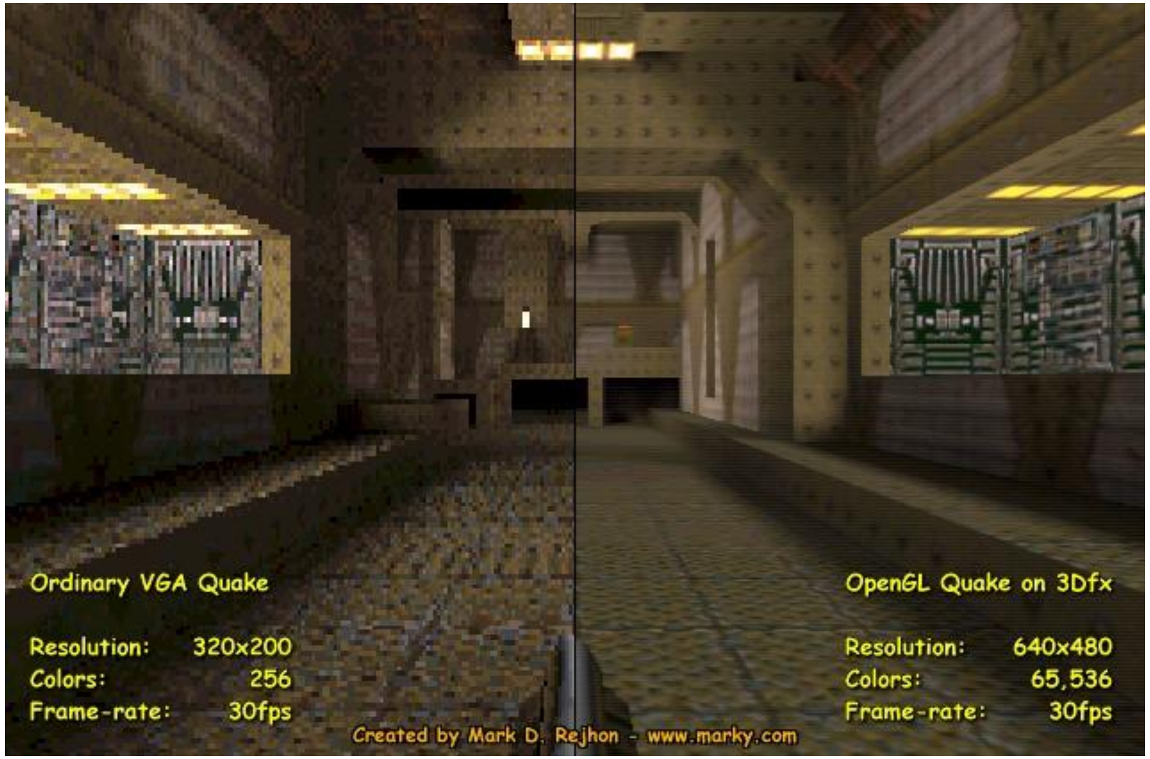


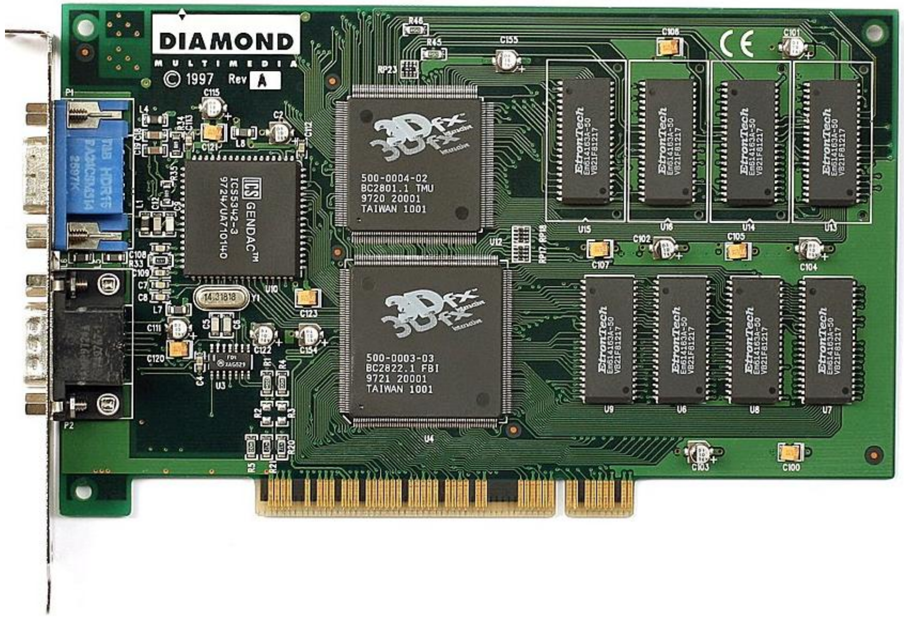
```
float Q_rsqrt( float number )
{
    const float x2 = number * 0.5F;
    const float threehalfs = 1.5F;

    union {
        float f;
        uint32_t i;
    } conv = {number}; // member 'f' set to value of 'number'.
    conv.i  = 0x5f3759df - ( conv.i >> 1 );
    conv.f  *= ( threehalfs - ( x2 * conv.f * conv.f ) );
    return conv.f;
}
```

Introduction of 3D Accelerator Cards

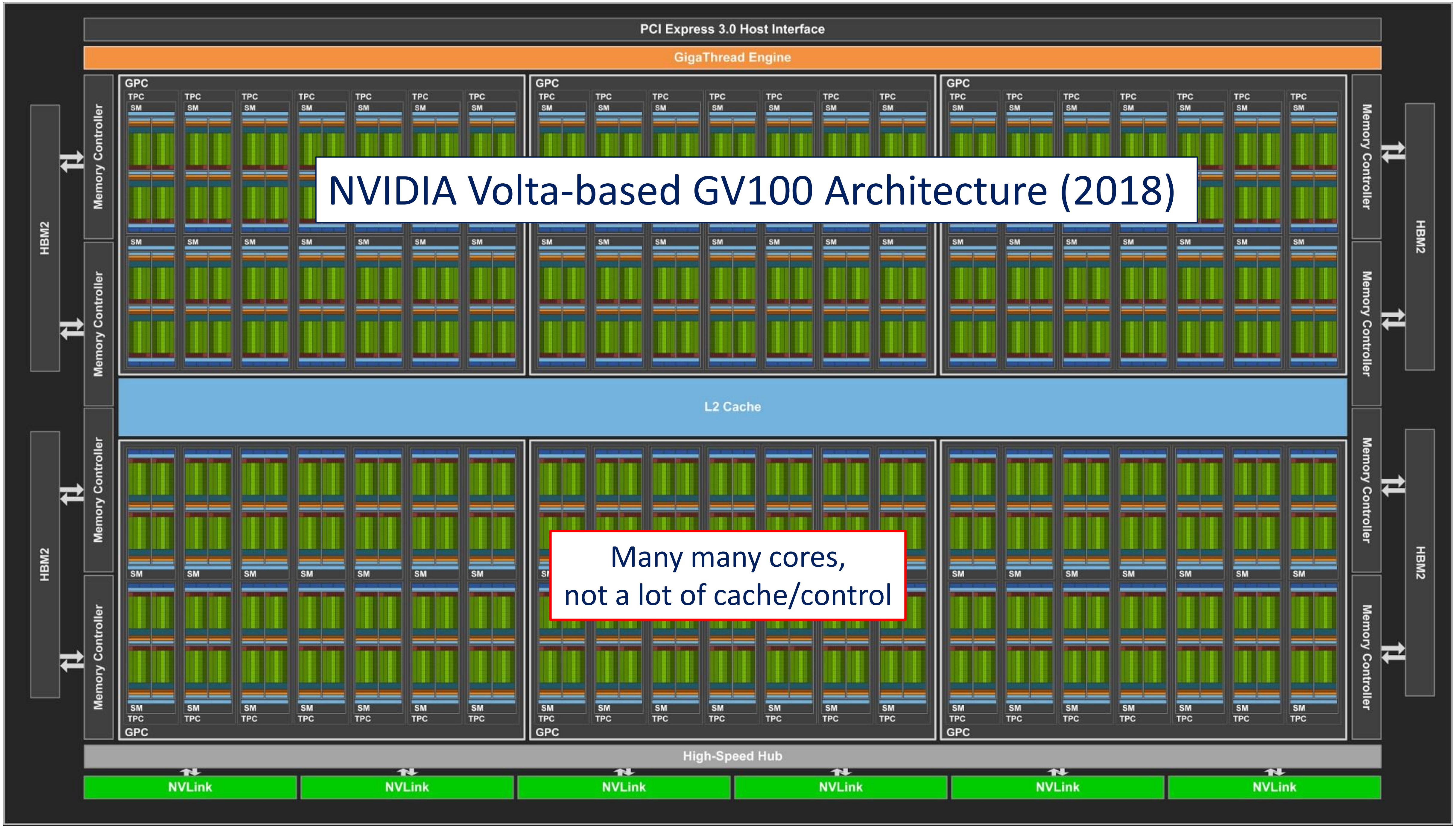
- ❑ Much of 3D processing is short algorithms repeated on a lot of data
  - pixels, polygons, textures, …
- ❑ Dedicated accelerators with simple, massively parallel computation





A Diamond Monster 3D, using the Voodoo chipset (1997)
(Konstantin Lanzet, Wikipedia)

NVIDIA Volta-based GV100 Architecture (2018)

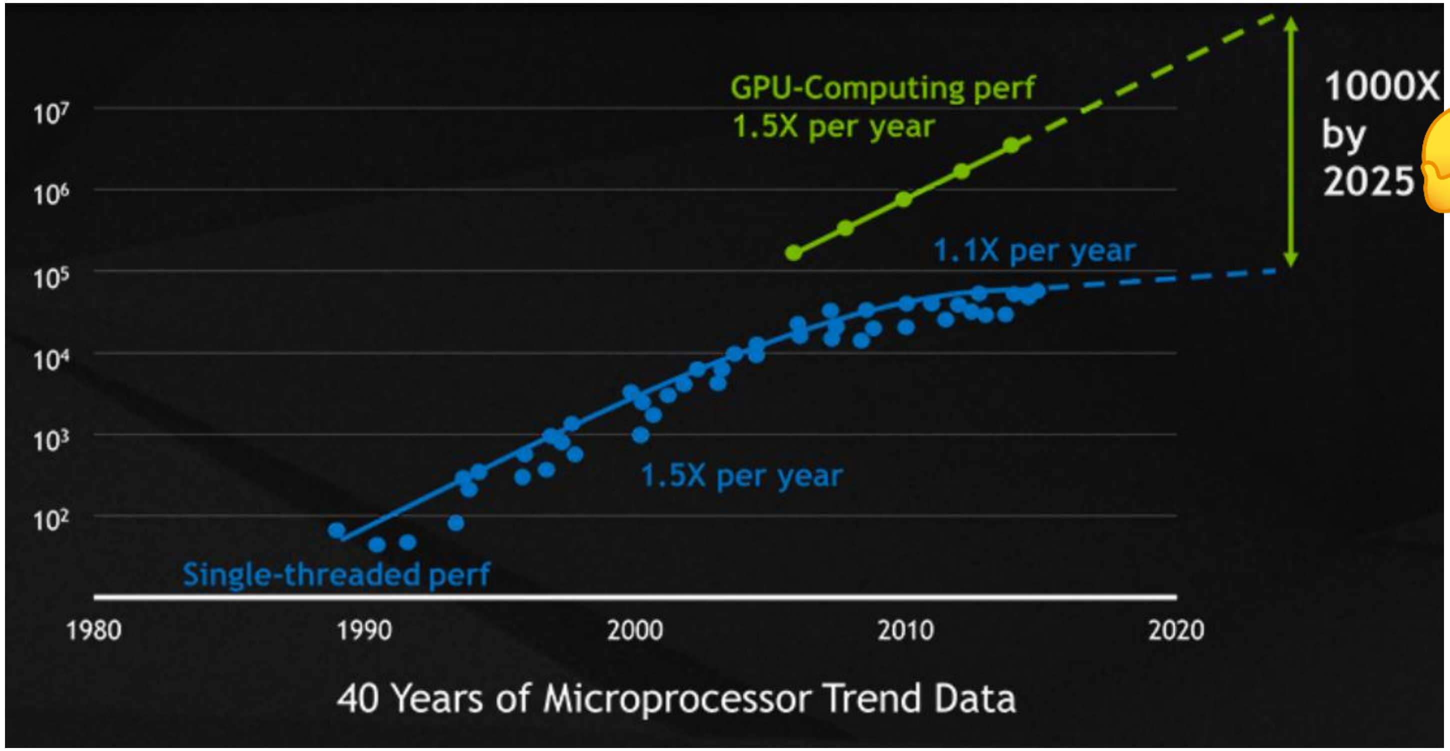


Many many cores,
not a lot of cache/control

Peak Performance vs. CPU

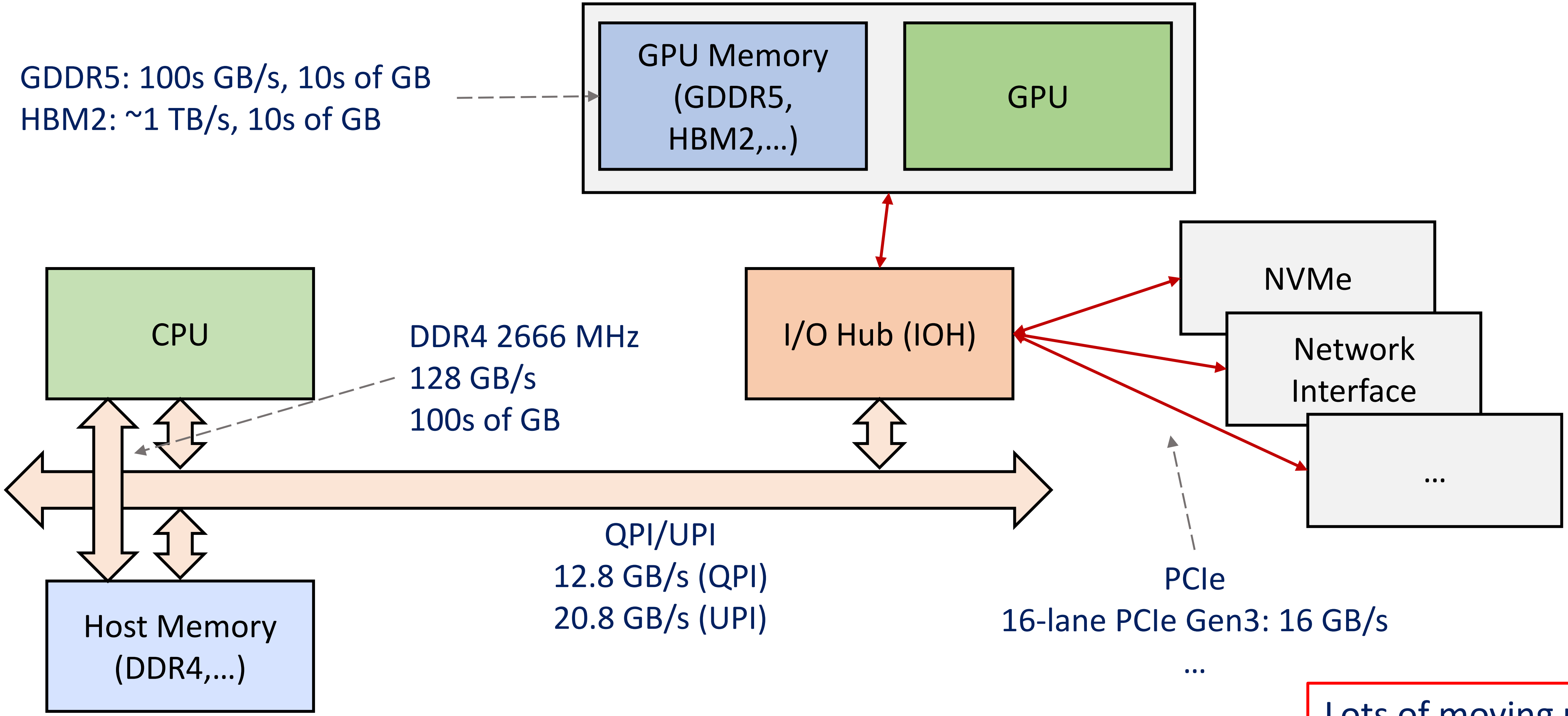
	Throughput	Power	Throughput/Power
Intel Skylake	128 SP GFLOPS/4 Cores	100+ Watts	~1 GFLOPS/Watt
NVIDIA V100	15 TFLOPS	200+ Watts	~75 GFLOPS/Watt

Also,



🤔

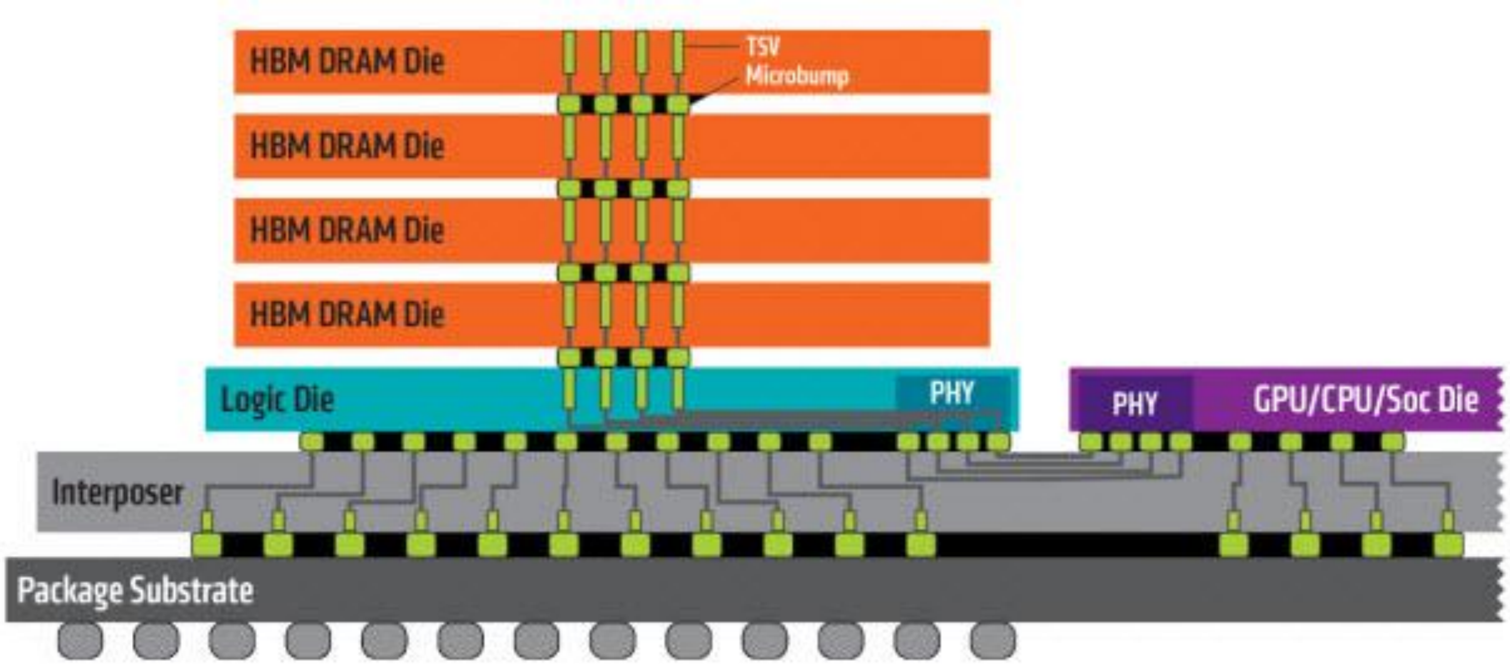
System Architecture Snapshot With a GPU (2019)



Lots of moving parts!

High-Performance Graphics Memory

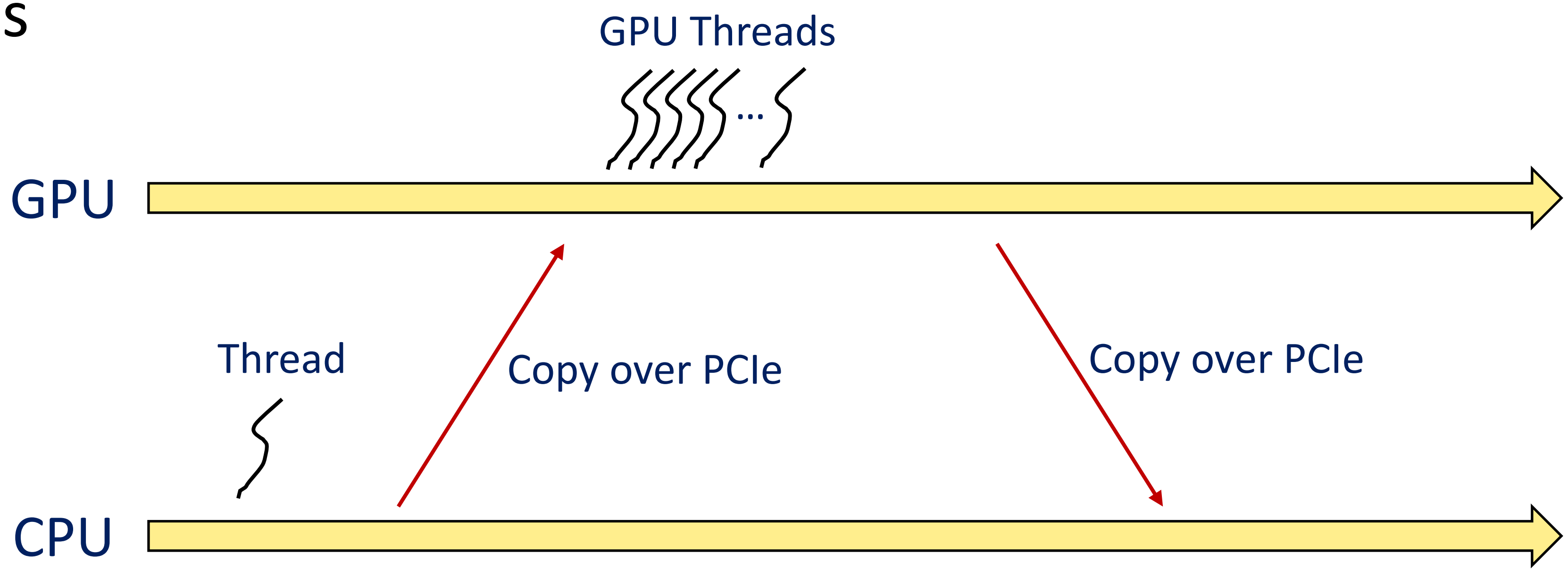
- ❑ Modern GPUs even employing 3D-stacked memory via silicon interposer
  - Very wide bus, very high bandwidth
  - e.g., HBM2 in Volta



Graphics Card Hub, "GDDR5 vs GDDR5X vs HBM vs HBM2 vs GDDR6 Memory Comparison," 2019

Massively Parallel Architecture For Massively Parallel Workloads!

- ❑ NVIDIA CUDA (Compute Uniform Device Architecture) – 2007
  - ○ A way to run custom programs on the massively parallel architecture!
- ❑ OpenCL specification released – 2008
- ❑ Both platforms expose synchronous execution of a massive number of threads

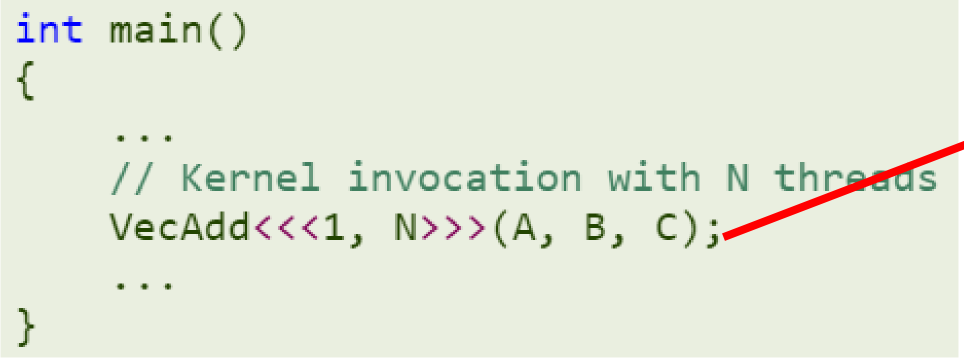


CUDA Execution Abstraction

- Block: Multi-dimensional array of threads
  - 1D, 2D, or 3D
  - Threads in a block can synchronize among themselves
  - Threads in a block can access shared memory
  - CUDA (Thread, Block) ~= OpenCL (Work item, Work group)
- Grid: Multi-dimensional array of blocks
  - 1D or 2D
  - Blocks in a grid can run in parallel, or sequentially
- Kernel execution issued in grid units
- Limited recursion (depth limit of 24 as of now)

Simple CUDA Example

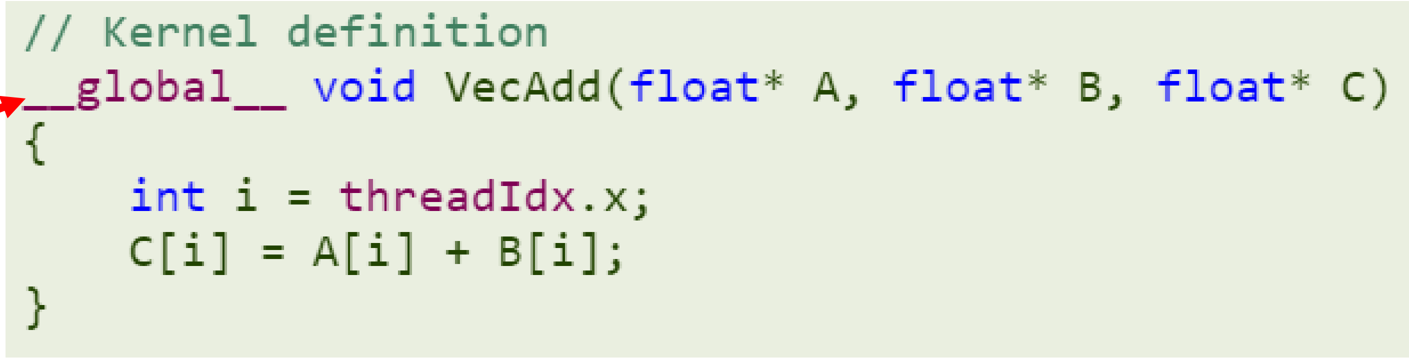
CPU side



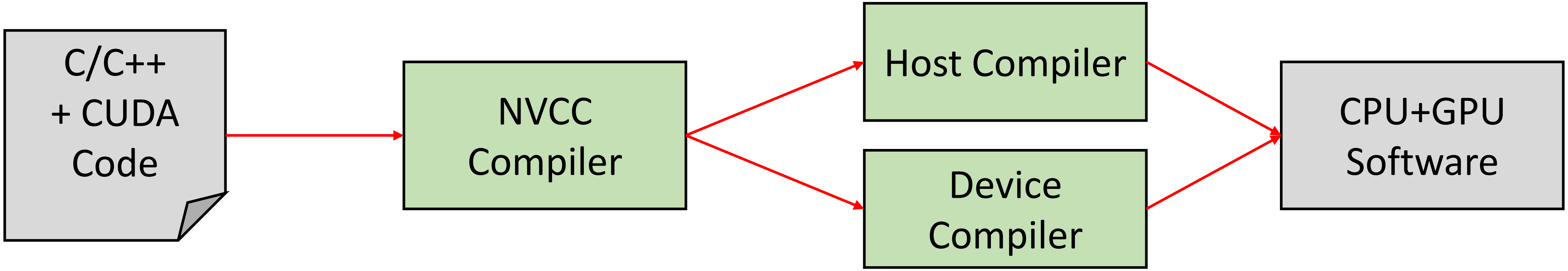
```
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Asynchronous call

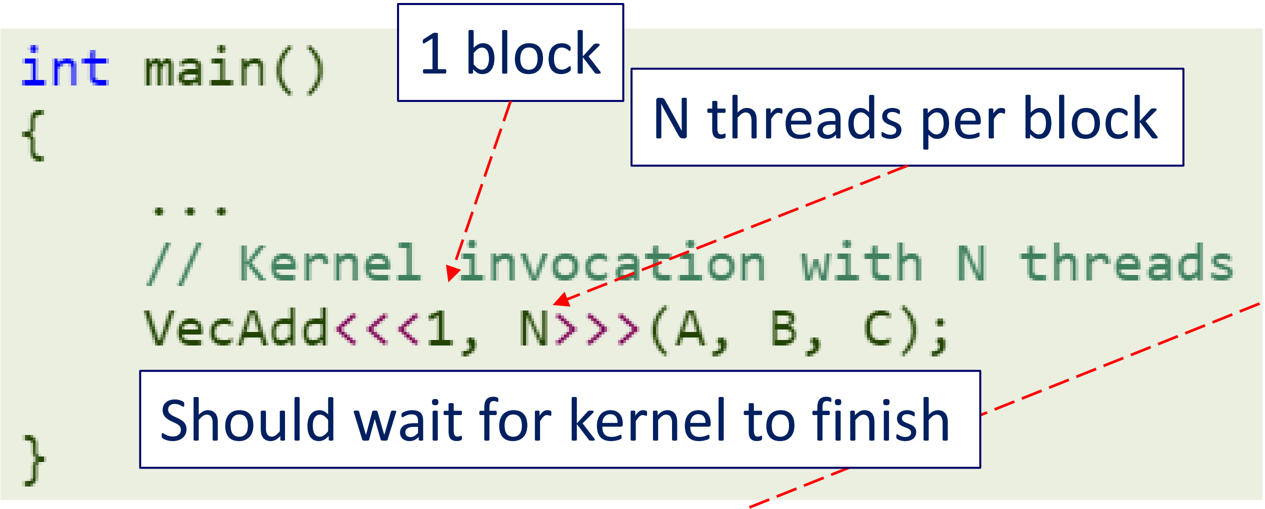
GPU side



```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```



Simple CUDA Example



```
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);

}
```

1 block

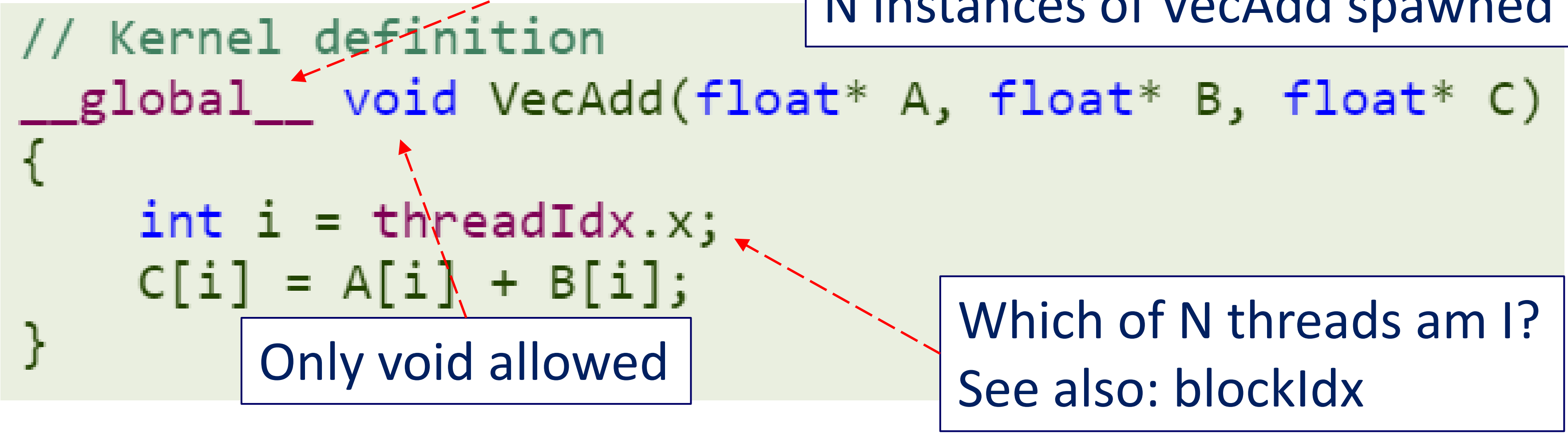
N threads per block

Should wait for kernel to finish

__global__:
In GPU, called from host/GPU

__device__:
In GPU, called from GPU

__host__:
In host, called from host



```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

N instances of VecAdd spawned in GPU

One function can be both

Only void allowed

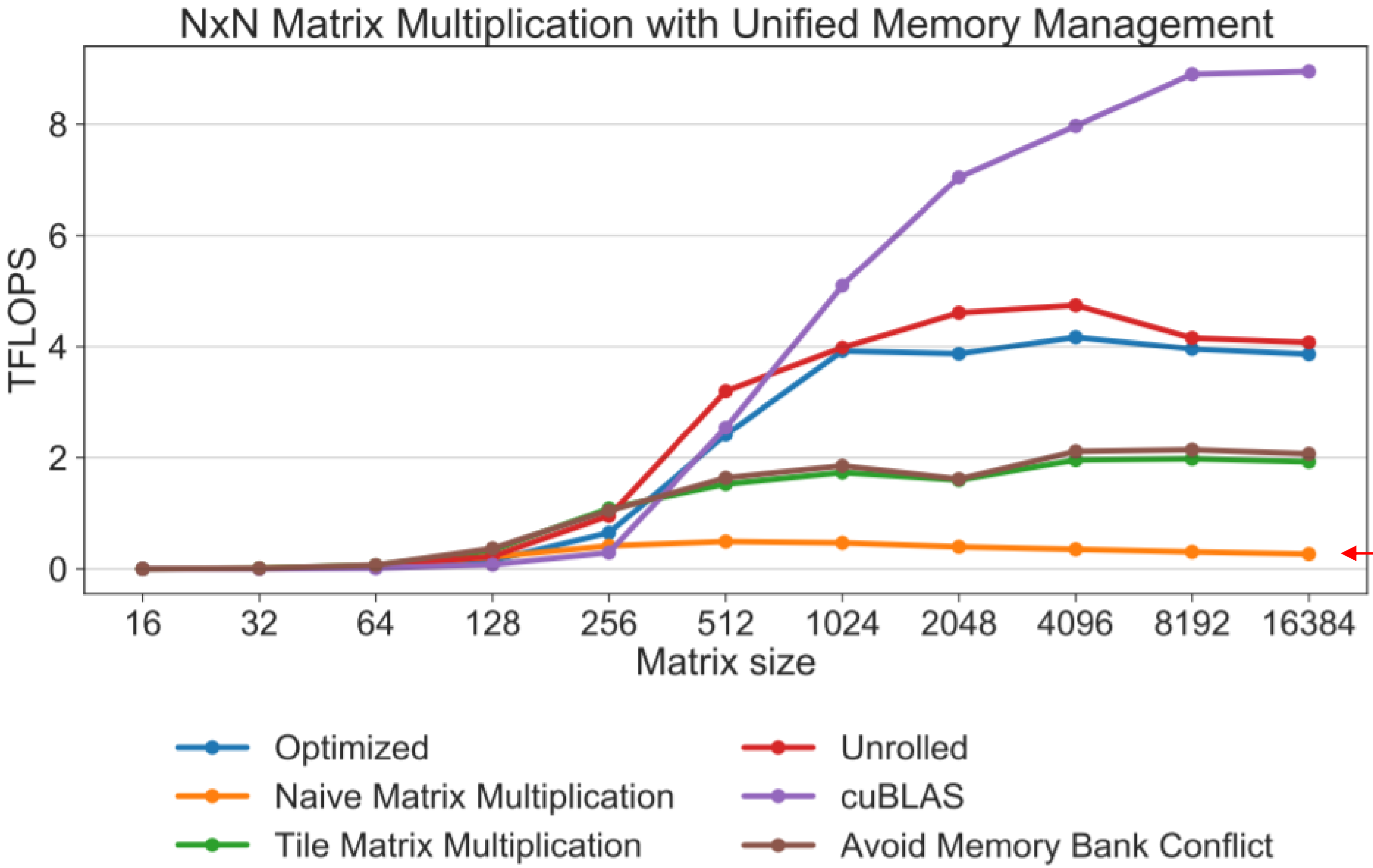
Which of N threads am I?
See also: blockIdx

More Complex Example: Picture Blurring

- ❑ Slides from NVIDIA/UIUC Accelerated Computing Teaching Kit
- ❑ Another end-to-end example
  https://devblogs.nvidia.com/even-easier-introduction-cuda/

- ❑ Great! Now we know how to use GPUs – Bye?

Matrix Multiplication Performance Engineering



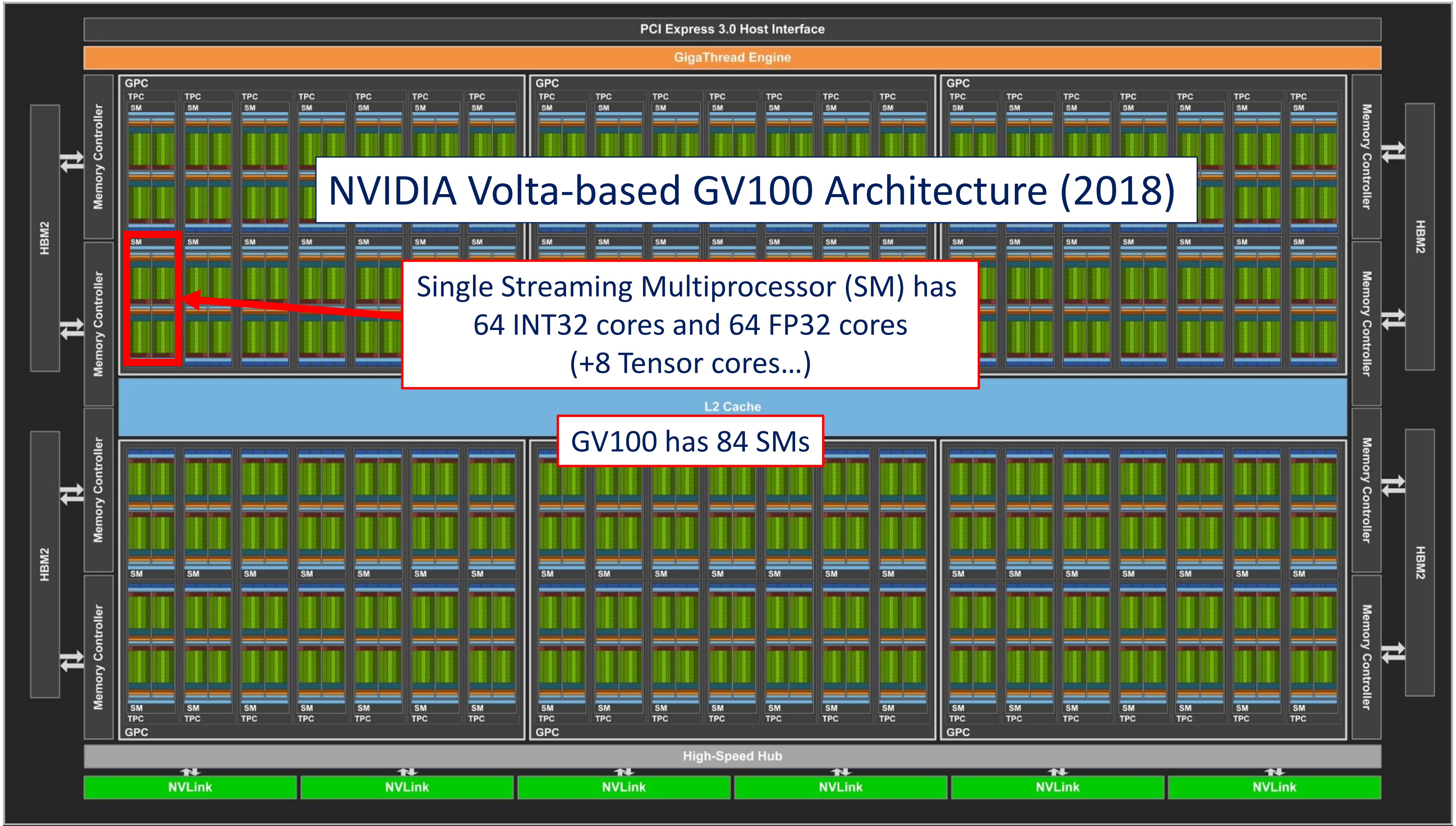
No faster than CPU

Results from NVIDIA P100

Coleman et. al., “Efficient CUDA,” 2017

Architecture knowledge is needed (again)

NVIDIA Volta-based GV100 Architecture (2018)



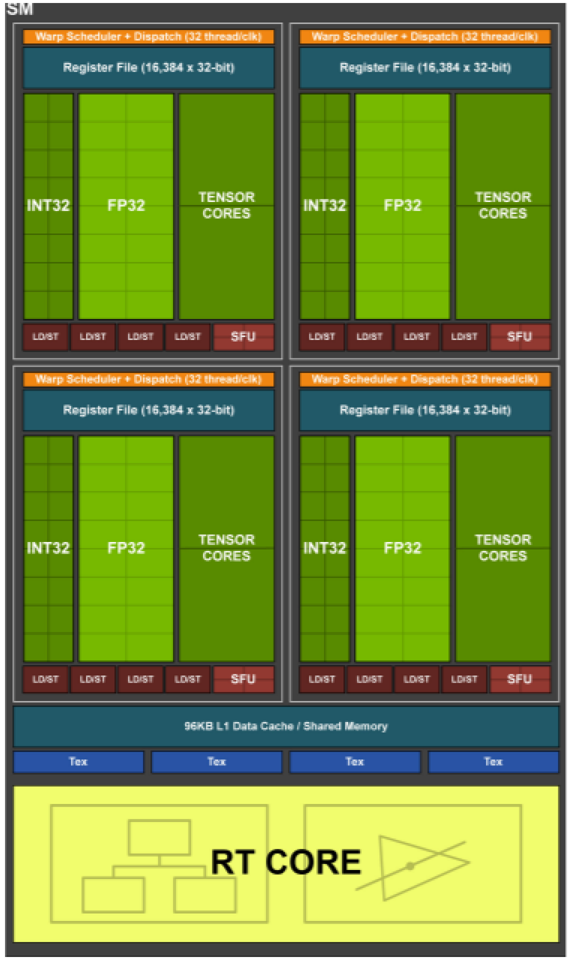
Single Streaming Multiprocessor (SM) has 64 INT32 cores and 64 FP32 cores (+8 Tensor cores…)

GV100 has 84 SMs

Volta Execution Architecture

- 64 INT32 Cores, 64 FP32 Cores, 4 Tensor Cores, Ray-tracing cores..
  - Specialization to make use of chip space…?
- Not much on-chip memory per thread
  - 96 KB Shared memory
  - 1024 Registers per FP32 core
- Hard limit on compute management
  - 32 blocks AND 2048 threads AND 1024 threads/block
  - e.g., 2 blocks with 1024 threads, or 4 blocks with 512 threads
  - Enough registers/shared memory for all threads must be available (all context is resident during execution)

More threads than cores – Threads interleaved to hide memory latency


SM
Warp Scheduler + Dispatch (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
TENSOR CORES
LD/ST
SFU
96KB L1 Data Cache / Shared Memory
Tex
RT CORE

Resource Balancing Details

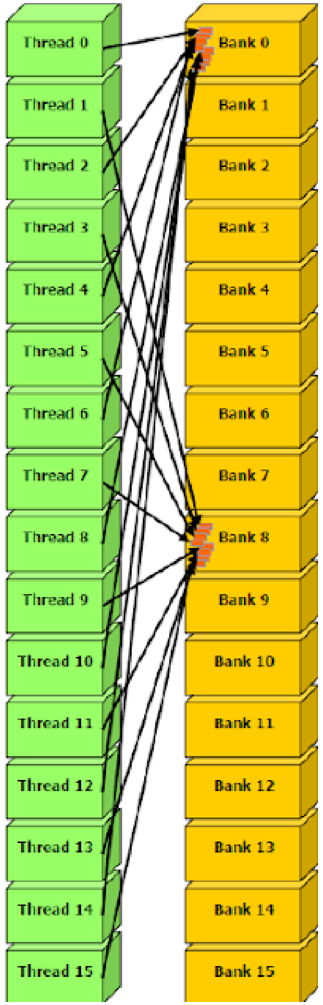
- ❑ How many threads in a block?
- ❑ Too small: 4x4 window == 16 threads
  - o 128 blocks to fill 2048 thread/SM
  - o SM only supports 32 blocks -> only 512 threads used
    - SM has only 64 cores… does it matter? Sometimes!
- ❑ Too large: 32x48 window == 1536 threads
  - o Threads do not fit in a block!
- ❑ Too large: 1024 threads using more than 64 registers
- ❑ Limitations vary across platforms (Fermi, Pascal, Volta, …)

Warp Scheduling Unit

- ❑ Threads in a block are executed in 32-thread “warp” unit
  - o Not part of language specs, just architecture specifics
  - o A warp is SIMD – Same PC, same instructions executed on every core
- ❑ What happens when there is a conditional statement?
  - o Prefix operations, or control divergence
  - o More on this later!
- ❑ Warps have been 32-threads so far, but may change in the future

Memory Architecture Caveats

- ❑ Shared memory peculiarities
  - Small amount (e.g., 96 KB/SM for Volta) shared across all threads
  - Organized into banks to distribute access
  - Bank conflicts can drastically lower performance
- ❑ Relatively slow global memory
  - Blocking, caching becomes important (again)
  - If not for performance, for power consumption…



8-way bank conflict
1/8 memory bandwidth