LAU
الجامعة اللبنانية الأميركية
Lebanese American University

CSC 447: Parallel Programming for Multi-Core and Cluster Systems

CUDA Thread Scheduling

Instructor: Haidar M. Harmanani

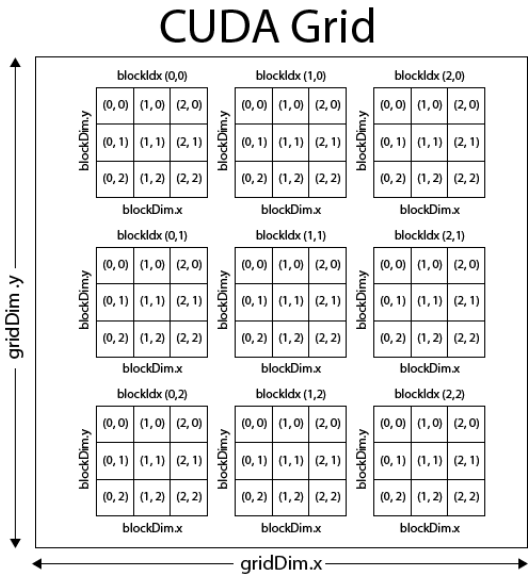
Spring 2017

Blocks, Grids, and Threads

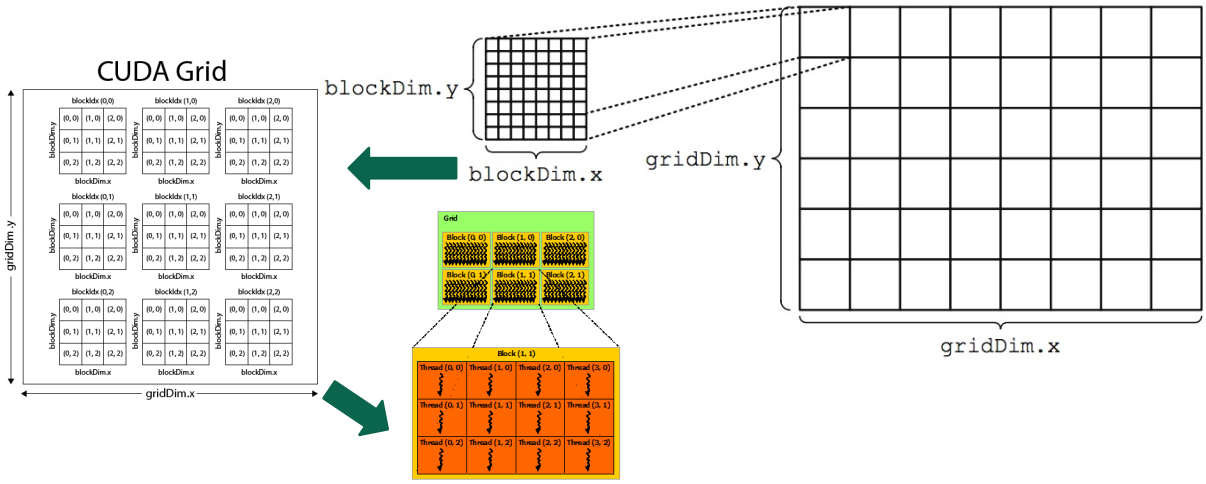
- When a kernel is launched, CUDA generates a grid of threads that are organized in a three-dimensional hierarchy
  - Each grid is organized into an array of thread blocks or *blocks*
  - Each block can contain up to 1,024 threads
  - Number of threads in a block is given in the `blockDim` variable
  - The dimension of thread blocks should be a multiple of 32
- Each thread in a block has a unique `threadIdx` value
  - Combine the **`threadIdx`** and **`blockIdx`** values to create a unique global index

Blocks, Grids, and Threads

- blockIdx: The block index within the grid
- gridDim: The dimensions of the grid
- blockDim: The dimensions of the block
- threadIdx: The thread index within the block.

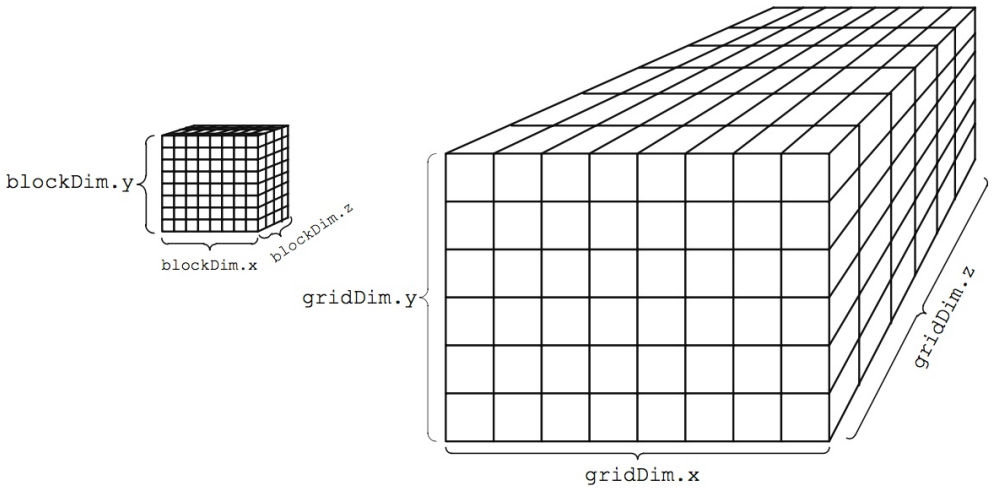


Blocks, Grids, and Threads



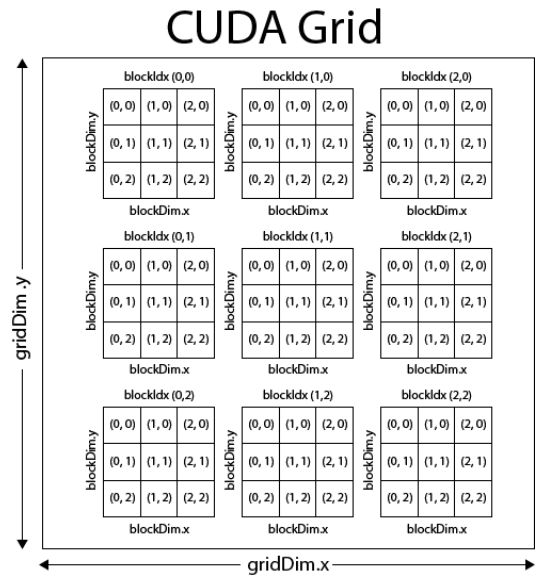
- Thread index = threadIdx.x + blockIdx.x * blockDim.x

Blocks, Grids, and Threads



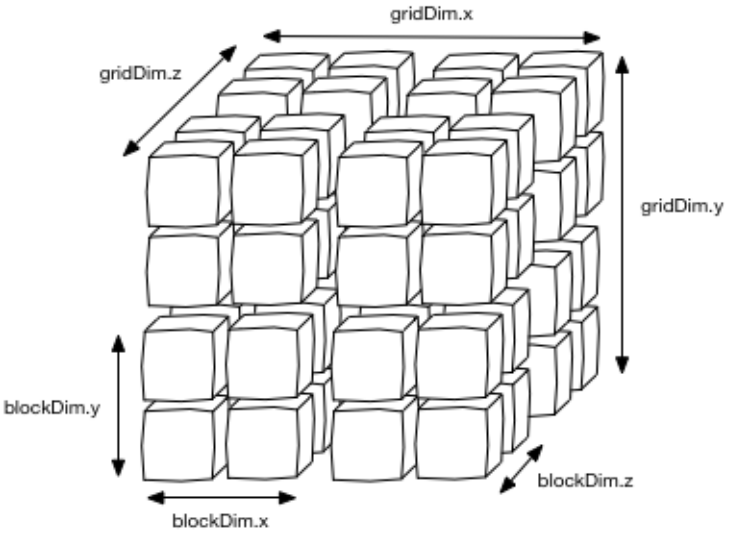
Global Thread IDs: 2D grid of 2D blocks

- tx = threadIdx.x
- ty = threadIdx.y
- bx = blockIdx.x
- by = blockIdx.y
- bw = blockDim.x
- bh = blockDim.y
- id_x = tx + bx * bw
- id_y = ty + by * bh



Global Thread IDs: 3D grid of 3D blocks

- tx = threadIdx.x
- ty = threadIdx.y
- tz = threadIdx.z
- bx = blockIdx.x
- by = blockIdx.y
- bz = blockIdx.y
- bw = blockDim.x
- bh = blockDim.y
- bd = blockDim.z
- id_x = tx + bx * bw
- id_y = ty + by * bh
- id_z = tz + bz * hd



Blocks, Grids, and Threads

- Each SM contains a number of CUDA cores
  - Fermi SM contains 32 processing cores
  - Pascal SM contains 64 processing cores
  - Maxwell contains 128 processing cores
  - Kepler SMX contains 192 processing cores
  - Fermi 32 and Tesla 8 CUDA cores into an SM
  - The GP100 SM is partitioned into two processing blocks, each having 32 single-precision CUDA Cores, an instruction buffer, a warp scheduler, 2 texture mapping units and 2 dispatch units.

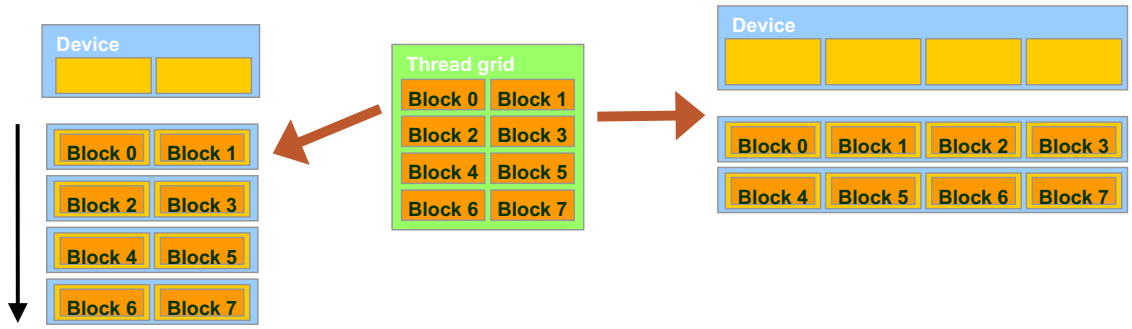
Blocks, Grids, and Threads

- 32 threads form a *warp*
  - Number of threads running concurrently on an MP
  - More about this later
- Instructions *are issued per warp*
  - It takes 4 clock cycles to issue a single instruction for the whole warp
- If an operand is not ready the warp will stall
- Threads in any given warp execute in lock-step, but to synchronise across warps, you need to use `__syncthreads()`

Blocks Must be Independent

- Any possible interleaving of blocks should be valid
  - presumed to run to completion without pre-emption
  - can run in any order
  - can run concurrently OR sequentially
- Blocks may coordinate but not synchronize
- Independence requirement gives *scalability*

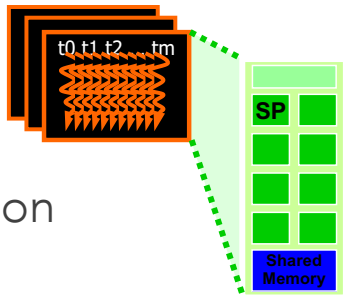
Transparent Scalability



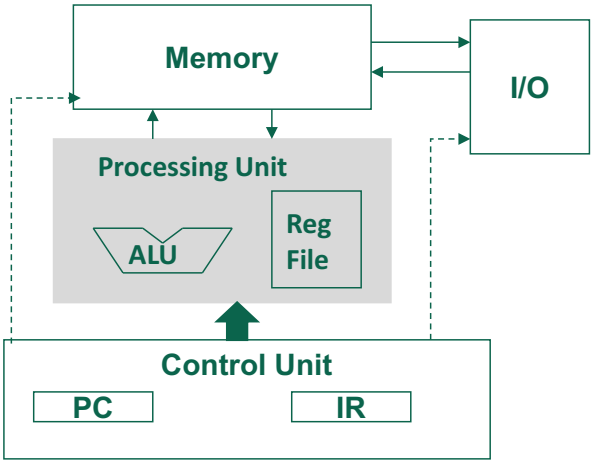
- Each block can execute in any order relative to others
  - Concurrently or sequentially
  - Facilitates scaling of the same code across many devices
- Hardware is free to assign blocks to any processor at any time
  - A kernel scales to any number of parallel processors

Example: Executing Thread Blocks

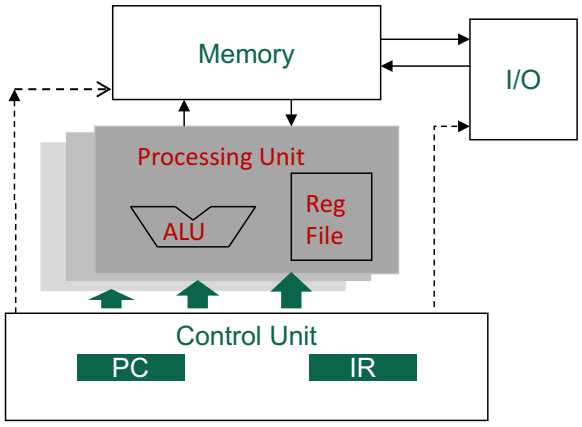
- Threads are assigned to *Streaming Multiprocessors (SM)* in block granularity
  - Up to 8 blocks to each SM as resource allows
  - Fermi SM can take up to 1536 threads
    - Could be 256 (threads/block) * 6 blocks
    - Or 512 (threads/block) * 3 blocks, etc.
- SM maintains thread/block idx #s
- SM manages/schedules thread execution



The Von-Neumann Model



The Von-Neumann Model with SIMD units



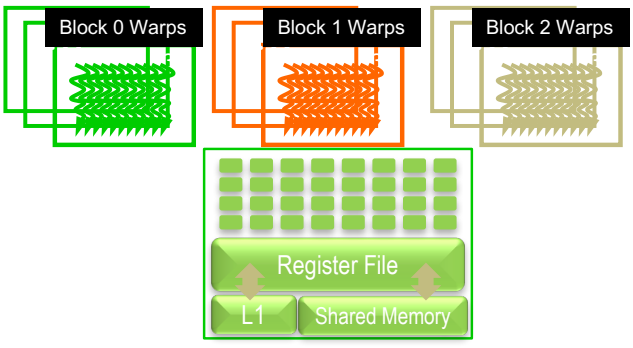
Single Instruction Multiple Data (SIMD)

Warps as Scheduling Units

- Each Block is executed as 32-thread Warps
  - An implementation decision, not part of the CUDA programming model
  - Warps are scheduling units in SM
  - Threads in a warp execute in SIMD
  - Future GPUs may have different number of threads in each warp

Warp Example

- If 3 blocks are assigned to an SM and each block has 256 threads, how many Warps are there in an SM?
  - Each Block is divided into 256/32 = 8 Warps
  - There are 8 * 3 = 24 Warps



Example: Thread Scheduling (Cont.)

- SM implements zero-overhead warp scheduling
  - Warps whose next instruction has its operands ready for consumption are eligible for execution
  - Eligible Warps are selected for execution based on a prioritized scheduling policy
  - All threads in a warp execute the same instruction when selected

Block Granularity Considerations

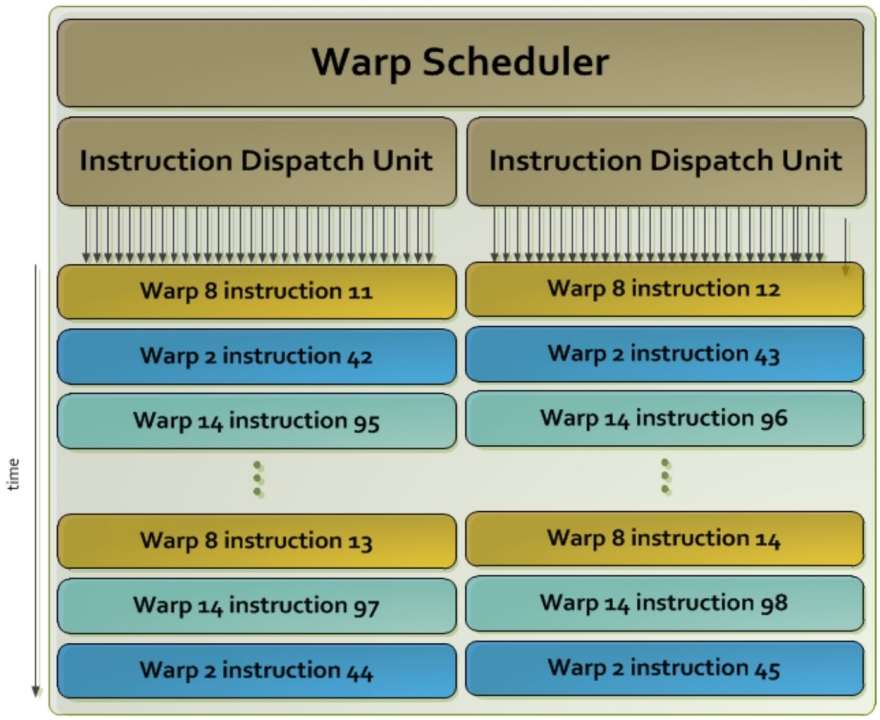
- For Matrix Multiplication using multiple blocks, should I use 8X8, 16X16 or 32X32 blocks for Fermi?
  - For 8X8, we have 64 threads per block.
    - We will need 1536/64 = 24 blocks to fully occupy an SM since each SM can take up to 1536 threads
    - However, each SM has only 8 Blocks, only 64x8 = 512 threads will go into each SM!
    - This means that the SM execution resources will likely be underutilized because there will be fewer warps to schedule around long latency operations.

Block Granularity Considerations

- For Matrix Multiplication using multiple blocks, should I use 8X8, 16X16 or 32X32 blocks for Fermi?
  - For 16X16, we have 256 threads per Block. Since each SM can take up to 1536 threads, it can take up to 6 Blocks and achieve full capacity unless other resource considerations overrule.
  - For 32X32, we would have 1024 threads per Block. Only one block can fit into an SM for Fermi. Using only 2/3 of the thread capacity of an SM.

More About Thread Scheduling

- When a warp is waiting on device memory, the scheduler switches to another ready warp, keeping as many cores busy as possible.
  - Because accessing device memory is so slow, the device *coalesces* global memory loads and stores issued by threads of a warp into as few transactions as possible
  - Because of coalescence, retrieval is optimal when neighboring threads (with consecutive indexes) access consecutive memory locations - i.e. with a stride of 1
  - A stride of 1 is not possible for indexing the higher dimensions of a dimensional array - shared memory is used to overcome this as there is no penalty for *strided* access to shared memory
  - Similarly, a structure consisting of arrays allows for efficient access, while an array of structures does not



Each Kepler SMX contains 4 Warp Schedulers, each with dual Instruction Dispatch Units. A single Warp Scheduler Unit is shown above.

Performance Tuning¶

- For optimal performance, the programmer has to juggle
  - finding enough parallelism to use all SMs
  - finding enough parallelism to keep all cores in an SM busy
  - optimizing use of registers and shared memory
  - optimizing device memory access for contiguous memory
  - organizing data or using the cache to optimize device memory access for contiguous memory