Thread and Memory Model

Tran Giang Son, tran-giang.son@usth.edu.vn

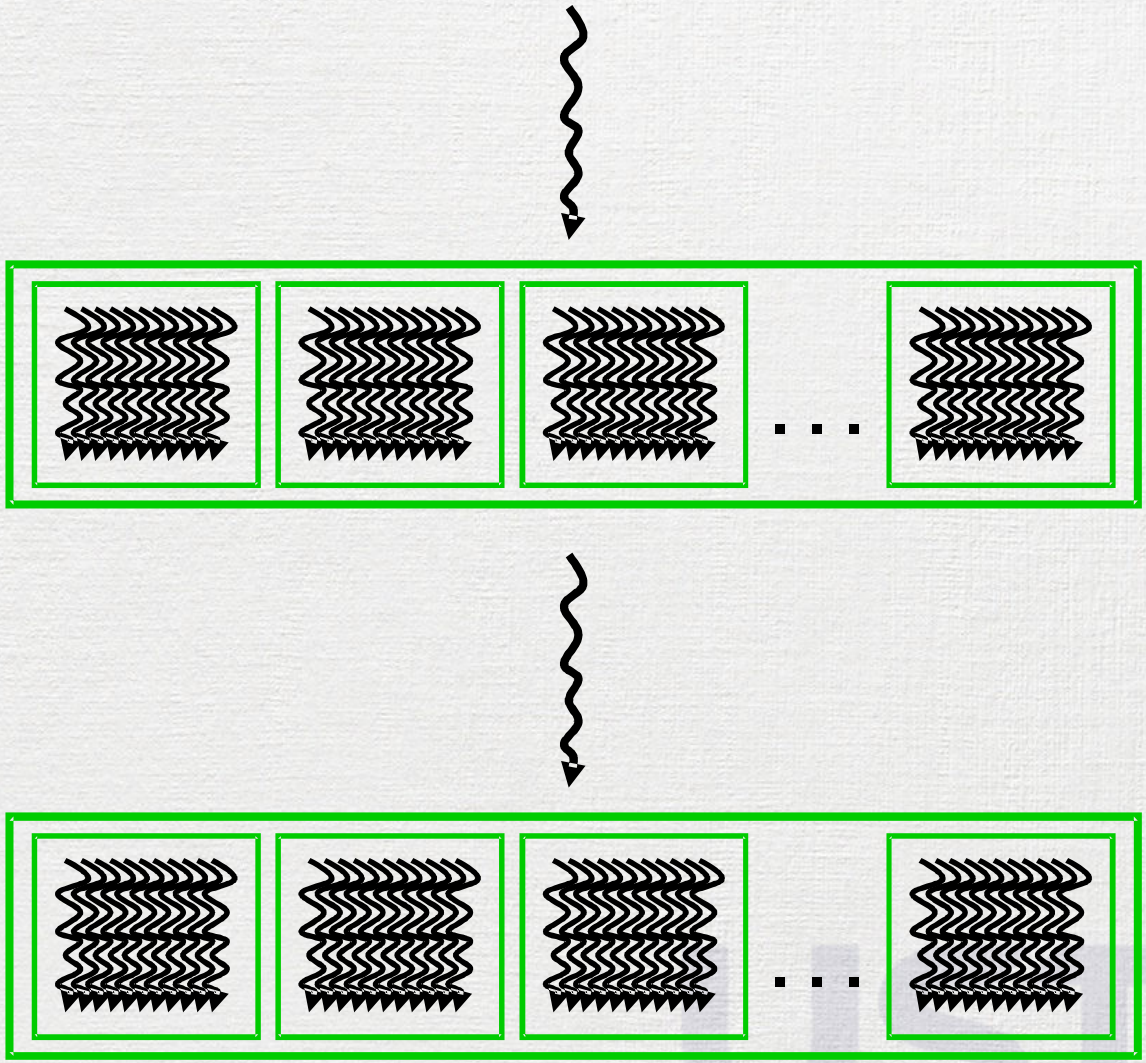
ICT Department, USTH

Thread Model

Thread

- What? a single sequential of execution
- SIMT on GPU
  - Same instruction
  - Same time
  - Different data
  - Natural for graphics and scientific computing
- A way to simplify core

Thread



Thread: Software View

- Thread: a single flow of kernel execution
- Block: a bunch of thread (1D, 2D, 3D)
  - `blockDim.x`, `blockDim.y`, `blockDim.z`
- Grid: a bunch of block (1D, 2D, 3D)
  - `gridDim.x`, `gridDim.y`, `gridDim.z`

Thread: Restrictions

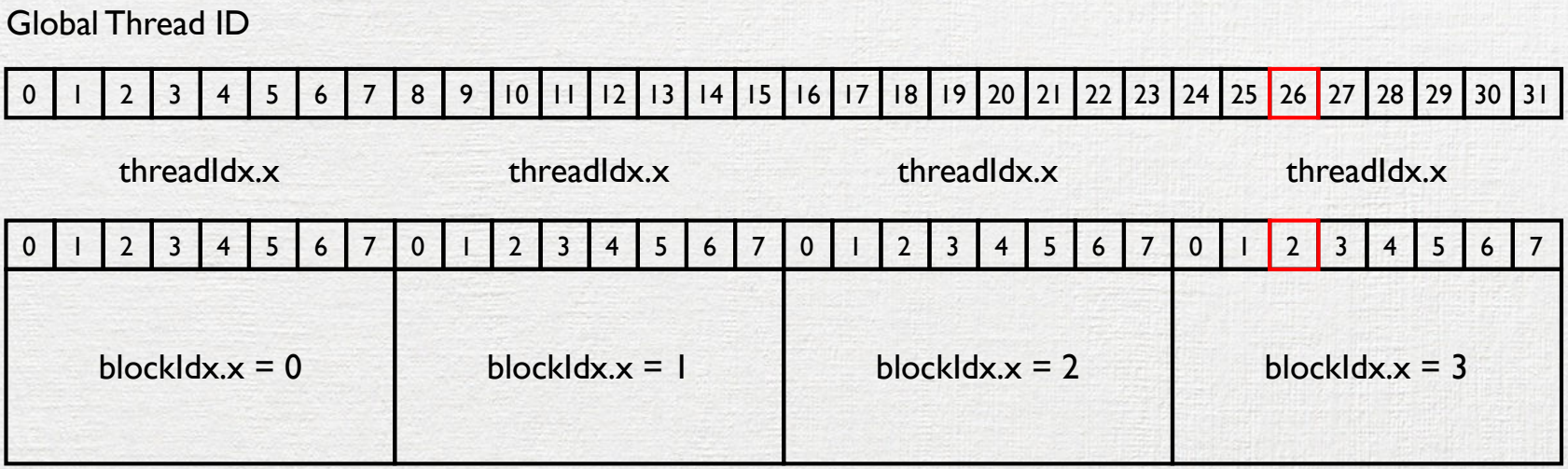
- Dimensions is fixed after kernel launch
- All blocks in a grid have the same dimension
- Block size and grid size are upper bounded

Thread: Restrictions

- Dimensions is fixed after kernel launch
- All blocks in a grid have the same dimension
- Block size and grid size are upper bounded

```
Maximum number of threads per multiprocessor:  2048
Maximum number of threads per block:           1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
```

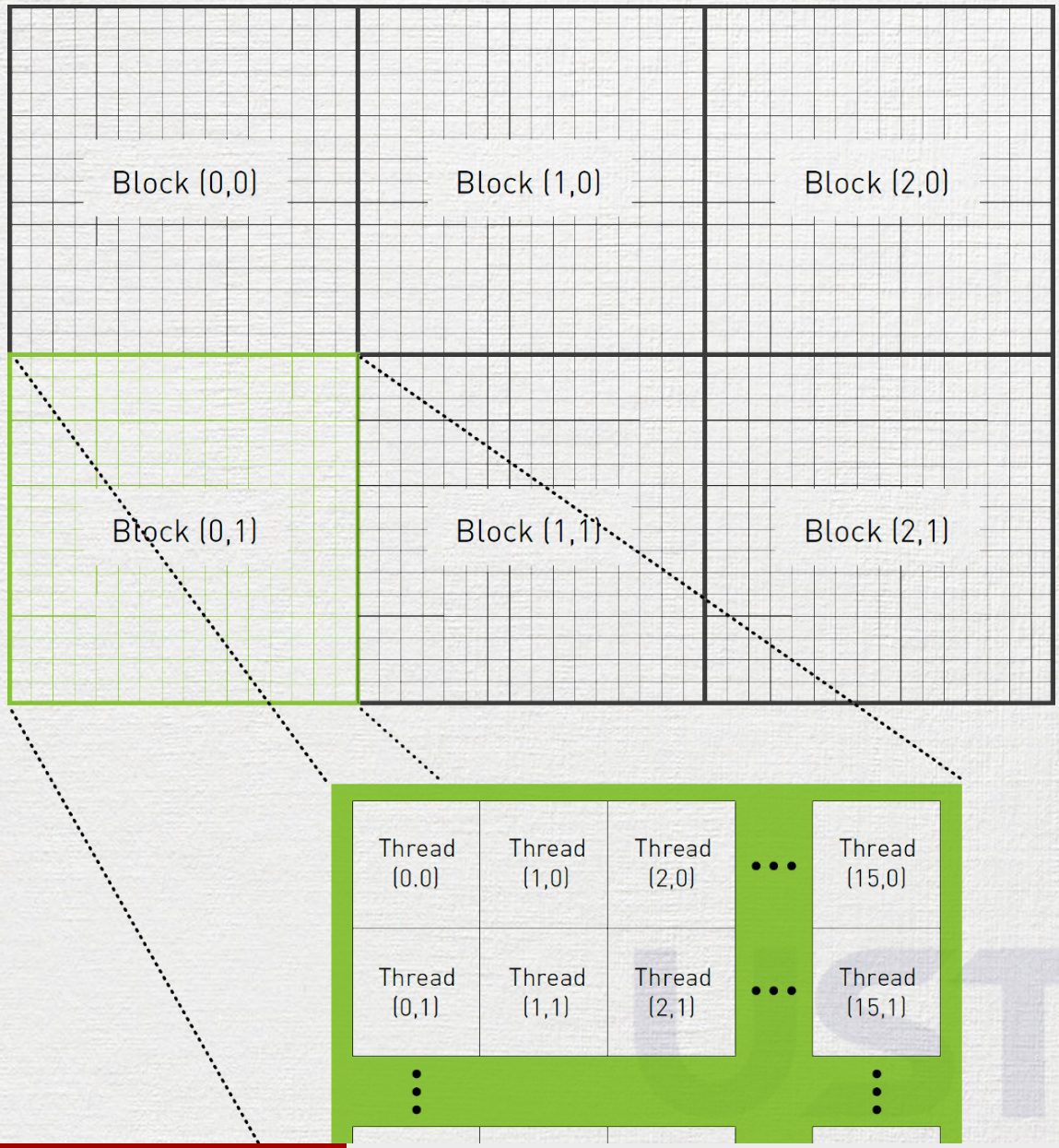
Thread: Software View




```
blockSize = 8

int globalThreadId = threadIdx.x + blockIdx.x * blockDim.x
```

Thread: Software View



Thread: Software View

- Where are we?
  - 1D: `x = threadIdx.x + blockIdx.x * blockDim.x`
  - 2D: `y = threadIdx.y + blockIdx.y * blockDim.y`
  - 3D: `z = threadIdx.z + blockIdx.z * blockDim.z`

Thread: Software View

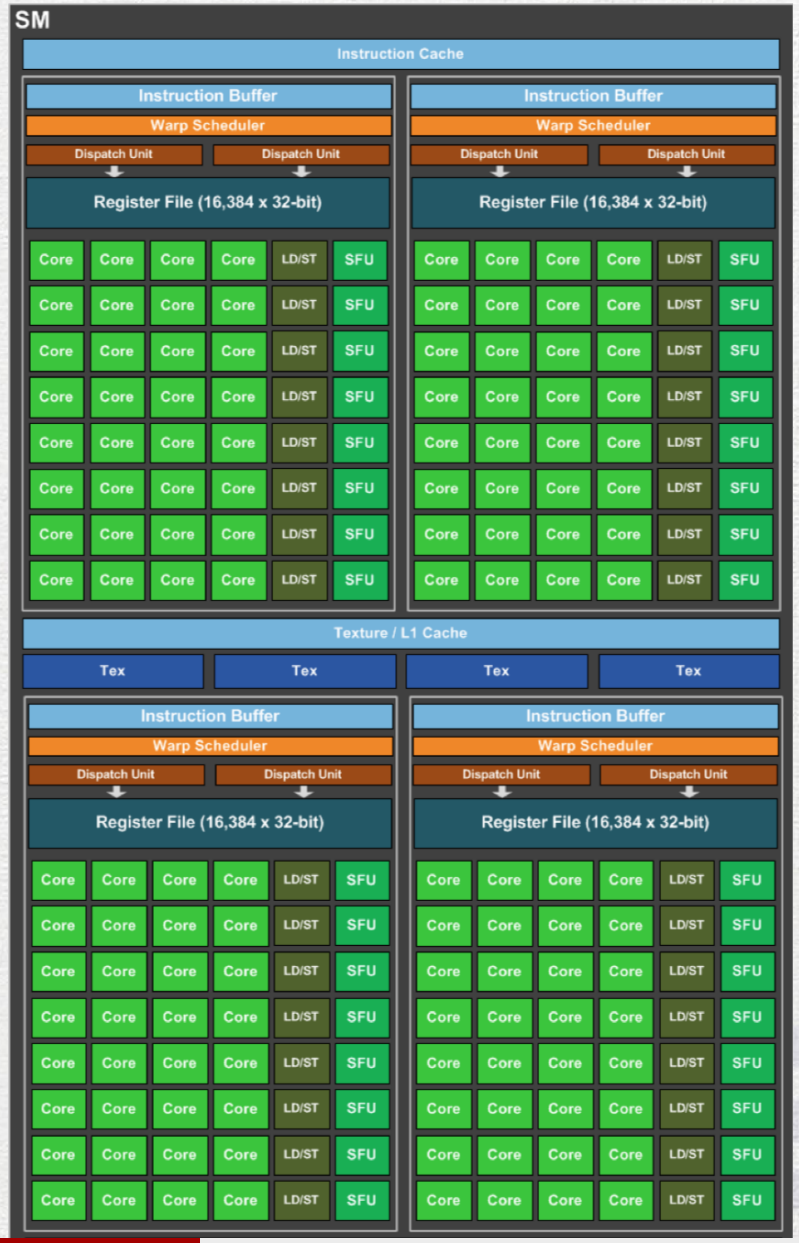
- Where are we?
  - 1D: `x = threadIdx.x + blockIdx.x * blockDim.x`
  - 2D: `y = threadIdx.y + blockIdx.y * blockDim.y`
  - 3D: `z = threadIdx.z + blockIdx.z * blockDim.z`
- How about `gridDim`?
  - Number of blocks in each dimension in the grid
  - Use case: 1D grid for a 2D image
    - Length of a row: `w = blockDim.x * gridDim.x`
    - Next row: `x += w`

Thread: Hardware View

- Streaming Processor (CUDA cores)
- Streaming Multiprocessor : A bunch of Streaming Processors plus some extra Special Function Units (sine/cosine/...)
- Graphics Processing Cluster : A bunch of Streaming Multiprocessors
- Many simple cores $\Rightarrow$ better performance

Thread: Hardware View

Thread: Hardware View



Thread: Assignment

- Each SM has “multiple of 32” cores

Thread: Assignment

- Each SM has “multiple of 32” cores
- Threads in SM execute in group of 32 threads
  - A group of 32 thread inside a SM is called « Warp »
  - Warp is unit of thread scheduling in SMs

Thread: Assignment

- Each SM has “multiple of 32” cores
- Threads in SM execute in group of 32 threads
  - A group of 32 thread inside a SM is called « Warp »
  - Warp is unit of thread scheduling in SMs
- Blocks are assigned to SMs into multiple of warps
  - Number of blocks per SM is constrained

Thread: Assignment

- Each SM has “multiple of 32” cores
- Threads in SM execute in group of 32 threads
  - A group of 32 thread inside a SM is called « Warp »
  - Warp is unit of thread scheduling in SMs
- Blocks are assigned to SMs into multiple of warps
  - Number of blocks per SM is constrained
- No specific mapping between thread and core

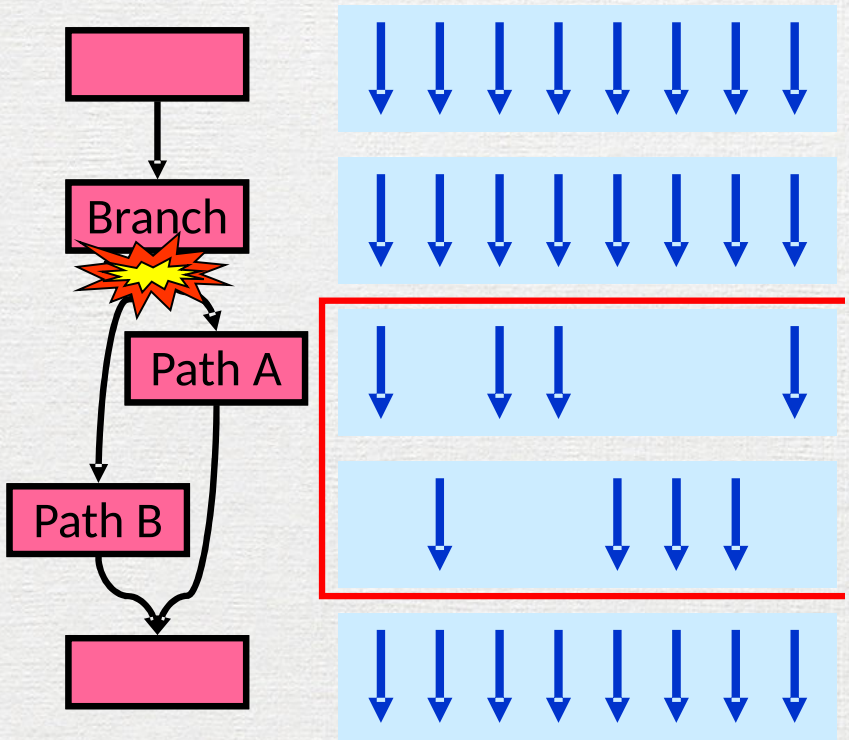
Thread: Assignment

- Each warp is executed in SIMD
  - All threads must execute same instruction at any time
- Fact
  - Not all warps are scheduled at anytime
  - Wait for data
  - Branch divergence

Thread: Assignment

- CUDA virtualizes the physical hardware
  - Thread : virtualized scalar processor
    - registers
    - PC
    - state
  - Block is a virtualized multiprocessor
    - threads
    - shared memory

Thread: Branch divergence



Thread: Branch divergence

- When?
  - Condition
- Divergence

```
if threadIdx.x > 2:
```

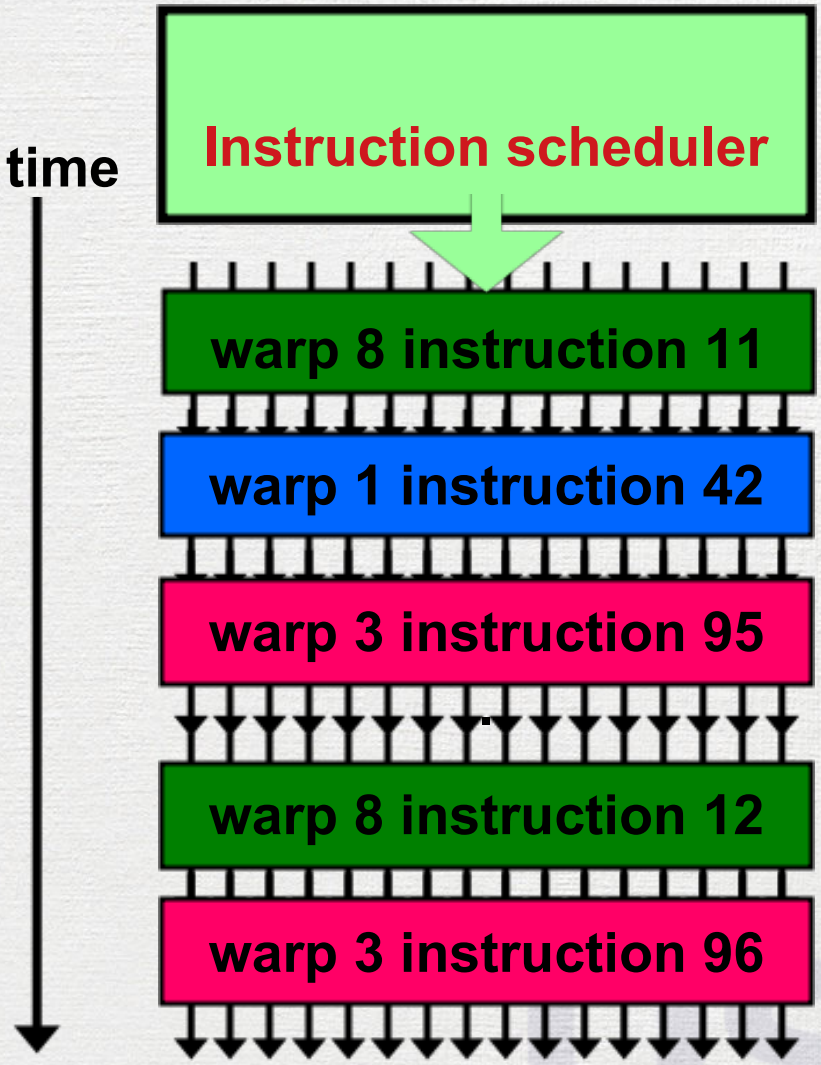
- No divergence

```
if threadIdx.x / WARP_SIZE > 2:
```

Thread: Latency Tolerance

- When a warp does something with high latency
  - Pause it
  - Schedule next warp
- No context switch
  - Large register file
  - No need to “switch” register content to memory
  - Zero overhead

Thread: Latency Tolerance



Thread: Latency Tolerance

- Latency tolerance relies on many warps
- Branch divergence does not affect GPU high throughput like CPU
- CPU focuses on low latency
  - Branch is important
  - Branch prediction is even more important

Block size in CUDA

- Previously, in launching kernel

```
kernelName[gridSize, blockSize](args...)
```

- Example

```
pixelCount = imageWidth * imageHeight
blockSize = 64
gridSize = pixelCount / blockSize
grayscale[gridSize, blockSize](devInput, devOutput)
```

- This is 1D kernel launch
  - `numBlock` is essentially `gridDim.x`

Block size in CUDA

- For 2D kernel launches
  - Grid size and block size are 2-dimensional tuples
- Launch a kernel with of 8×8 blocks, each block has 32×32 threads

```
gridSize = (8, 8)
blockSize = (32, 32)
grayscale[gridSize, blockSize](devInput, devOutput)
```

Labwork & Exercises 4: Threads

- Copy labwork 3 code to labwork 4
- Improve labwork 4 code to use 2D blocks
- Use `time.time()` to measure speedup
- Write a report (in LaTeX)
  - Name it « Report.4.threads.tex »
  - Explain how you improve the labwork
  - Try experimenting with different 2D block size values
  - Plot a graph of block size vs speedup
  - Compare speedup with previous 1D grid
  - Answer the questions in the upcoming slides, explain why
- Push the report and your code to your forked repository

Thread: Exercises 1

Consider a GPU having the following specs (maximum numbers):

- 512 threads/block
- 1024 threads/SM
- 8 blocks/SM
- 32 threads/warp

What is the best configuration for thread blocks to implement grayscaling?

- 8×8
- 16×16
- 32×32

Thread: Exercises 2

Consider a device SM that can take max

- 1,536 threads
- 4 blocks

Which of the following block configs would result in the most number of threads in the SM?

- 128 threads/blk
- 256 threads/blk
- 512 threads/blk
- 1,024 threads/blk

Thread: Exercises 3

Consider a vector addition problem

- Vector length is 2,000
- Each thread produces one output
- Block size 512 threads.

How many threads will be in the grid?