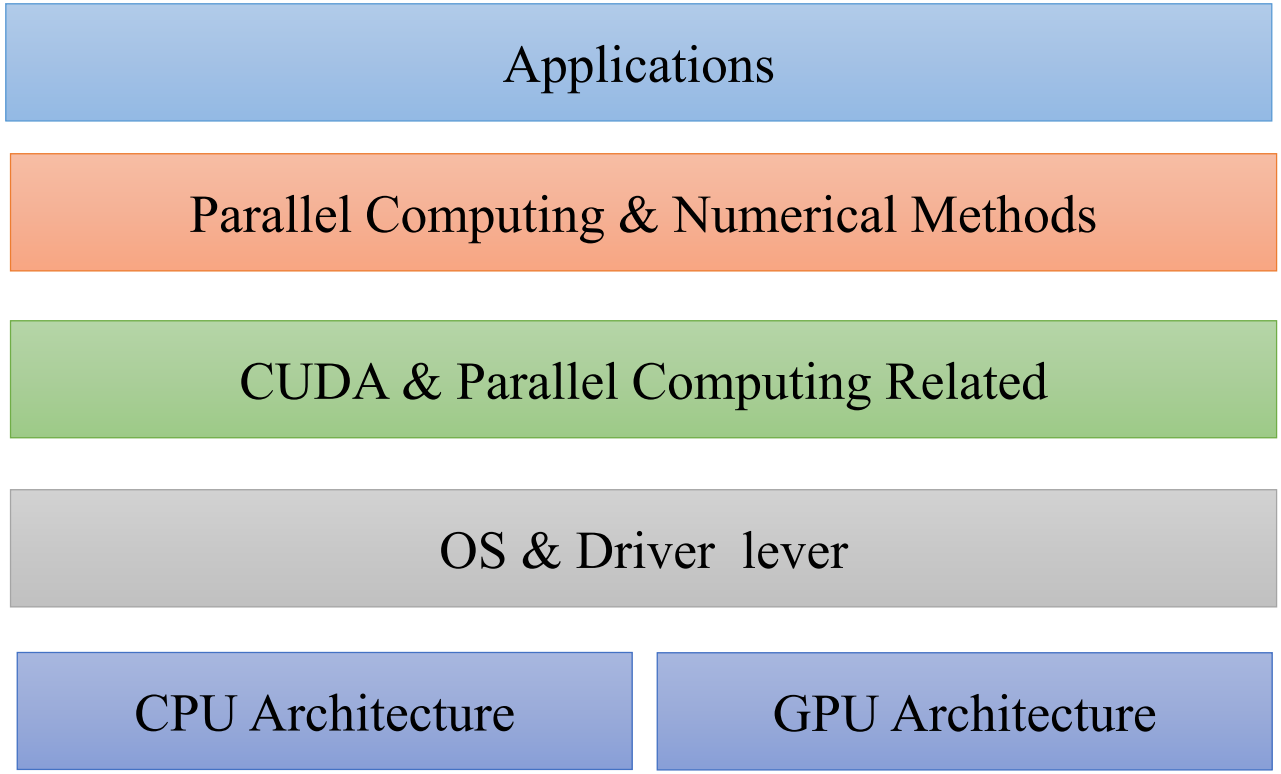
GPU/CUDA workshop

SMP

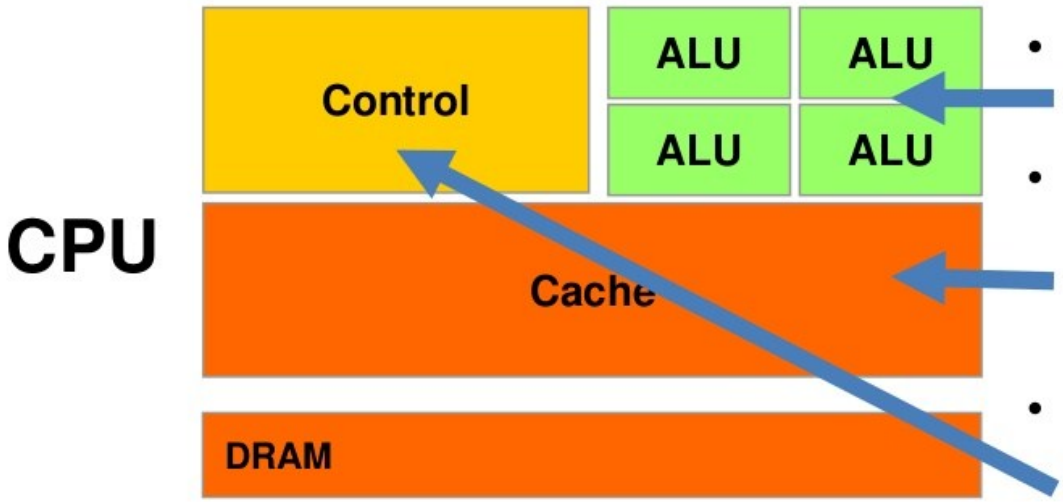
2018/10/26

GPU


Applications
Parallel Computing & Numerical Methods
CUDA & Parallel Computing Related
OS & Driver lever
CPU Architecture
GPU Architecture

CPU vs GPU

CPUs: Latency Oriented Design

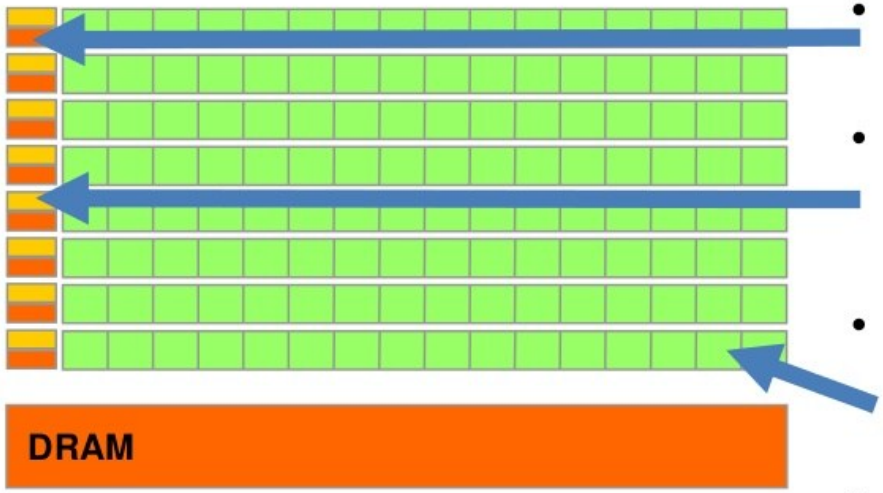


- Powerful ALU
  - Reduced operation latency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency

CPU vs GPU

GPUs: Throughput Oriented Design

GPU



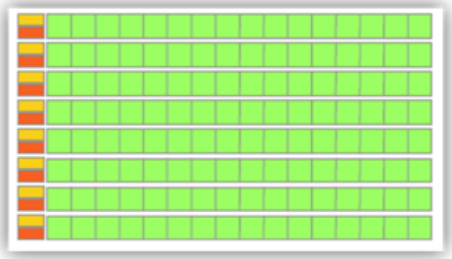
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies

CPU vs GPU

CPU

* Low compute density
* Complex control logic
* Large caches (L1$/L2$, etc.)
* Optimized for serial operations
  - Fewer execution units (ALUs)
  - Higher clock speeds
* Shallow pipelines (<30 stages)
* Low Latency Tolerance
* Newer CPUs have more parallelism

GPU



* High compute density
* High Computations per Memory Access
* Built for parallel operations
  - Many parallel execution units (ALUs)
  - Graphics is the best known case of parallelism
* Deep pipelines (hundreds of stages)
* High Throughput
* High Latency Tolerance
* Newer GPUs:
  - Better flow control logic (becoming more CPU-like)
  - Scatter/Gather Memory Access
  - Don't have one-way pipelines anymore

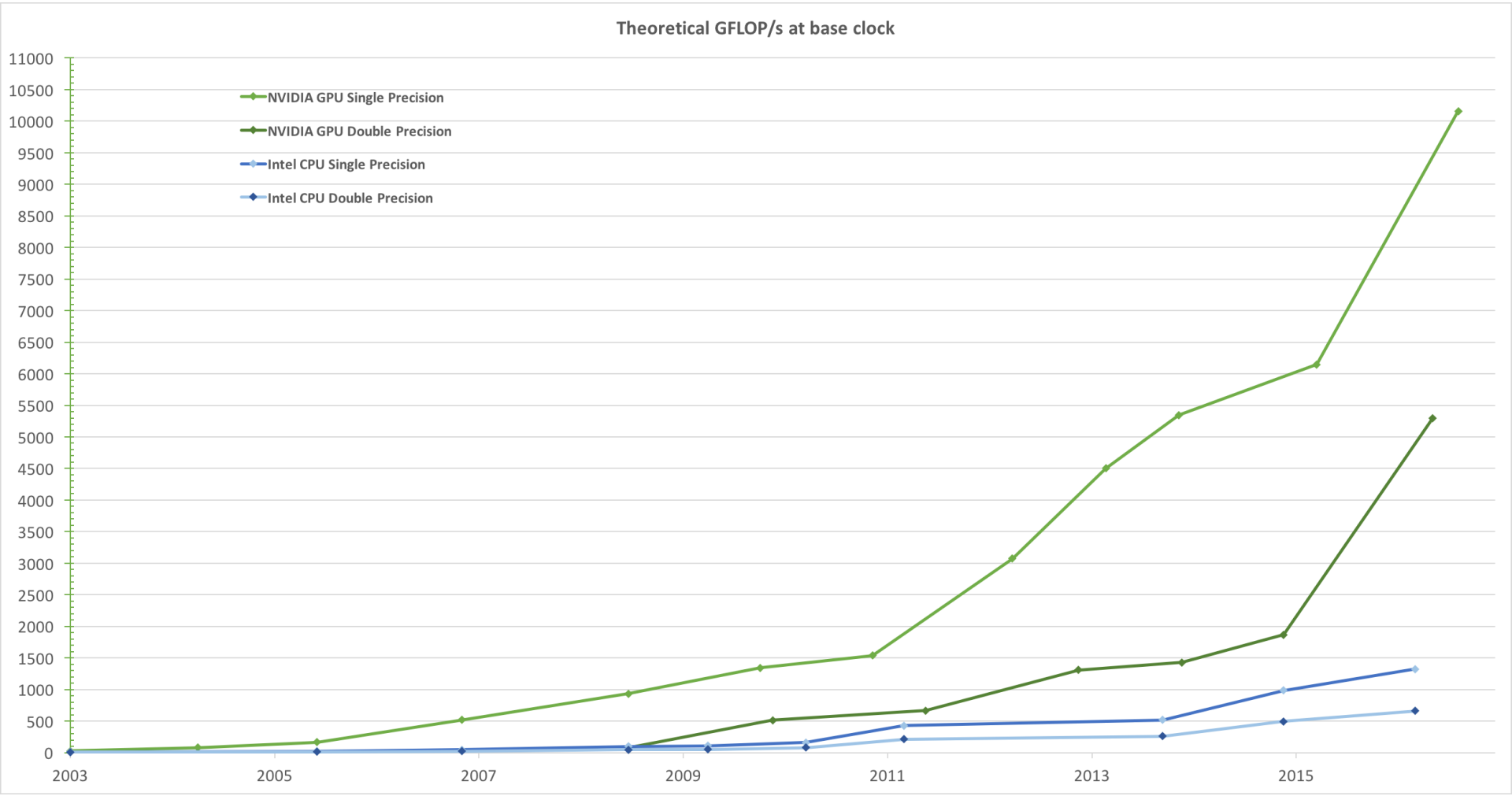
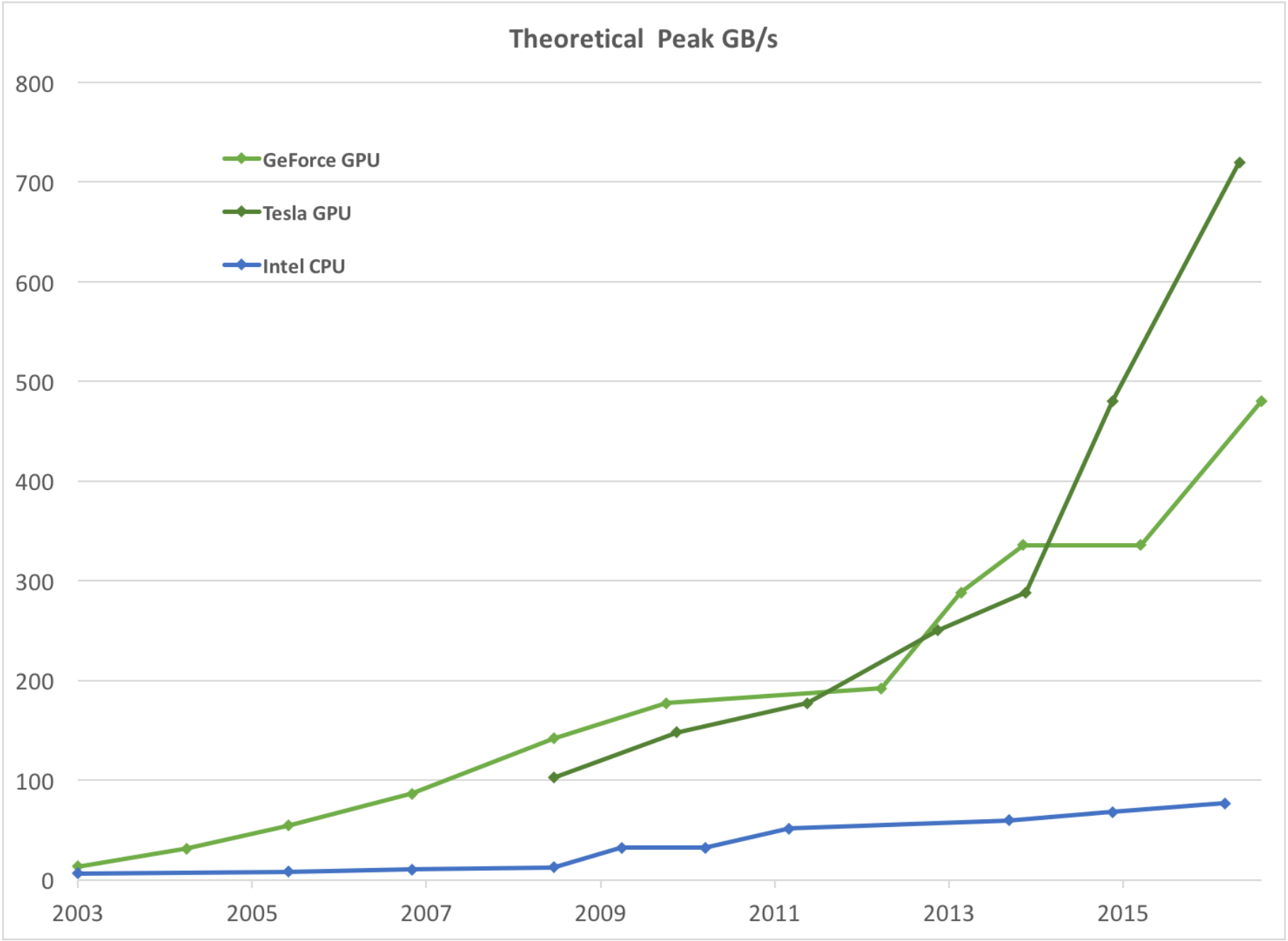


Figure 1. Floating-Point Operations per Second for the CPU and GPU

Figure 2. Memory Bandwidth for the CPU and GPU



cat /proc/cpuinfo

Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz

/opt/cuda-samples/NVIDIA_CUDA-7.5_Samples/bin/x86_64/linux/release

```
Device 0: GeForce GTX 980 Ti
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  33554432                     12161.3

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  33554432                     12199.0

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  33554432                     211199.3

Result = PASS
```

/usr/local/cuda/samples/1_Utilities/bandwidthTest

```
 Device 0: TITAN Xp
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     11607.8

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     12850.3

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     413598.7

Result = PASS
```

Flynn matrix

- The matrix below defines the 4 possible classifications according to Flynn

SISD Single Instruction, Sigle Data	SIMD Single Instruction, Multiple Data
MISD Multiple Instruction, Sigle Data	MIMD Multiple Instruction, Multiple Data

TTCAL

```
#!/bin/sh


for((i=221;i<=230;++i))
do
      # mkdir /udisk/dashenb2018xjc/zt_dashen/${i}
        ((n=360))
        ((m=(${i}-221)))

        echo $i

        for((j=0;j<1;++j))
        do
                ((is1=n*${j}+n*${m}))
                ((is2=n*${j}+${n}-1+n*${m}))


                echo $is1 $is2
        #       ssh -X gnode${i} "uname -a"
                ssh yc@gnode${i} ttqwfc input=/udisk/DQ_int/
=200.0 dxs=200.0 dxt=100.0 nxt=201 dyt=100.0 nyt=201 dzt=20

        done

done
~
```

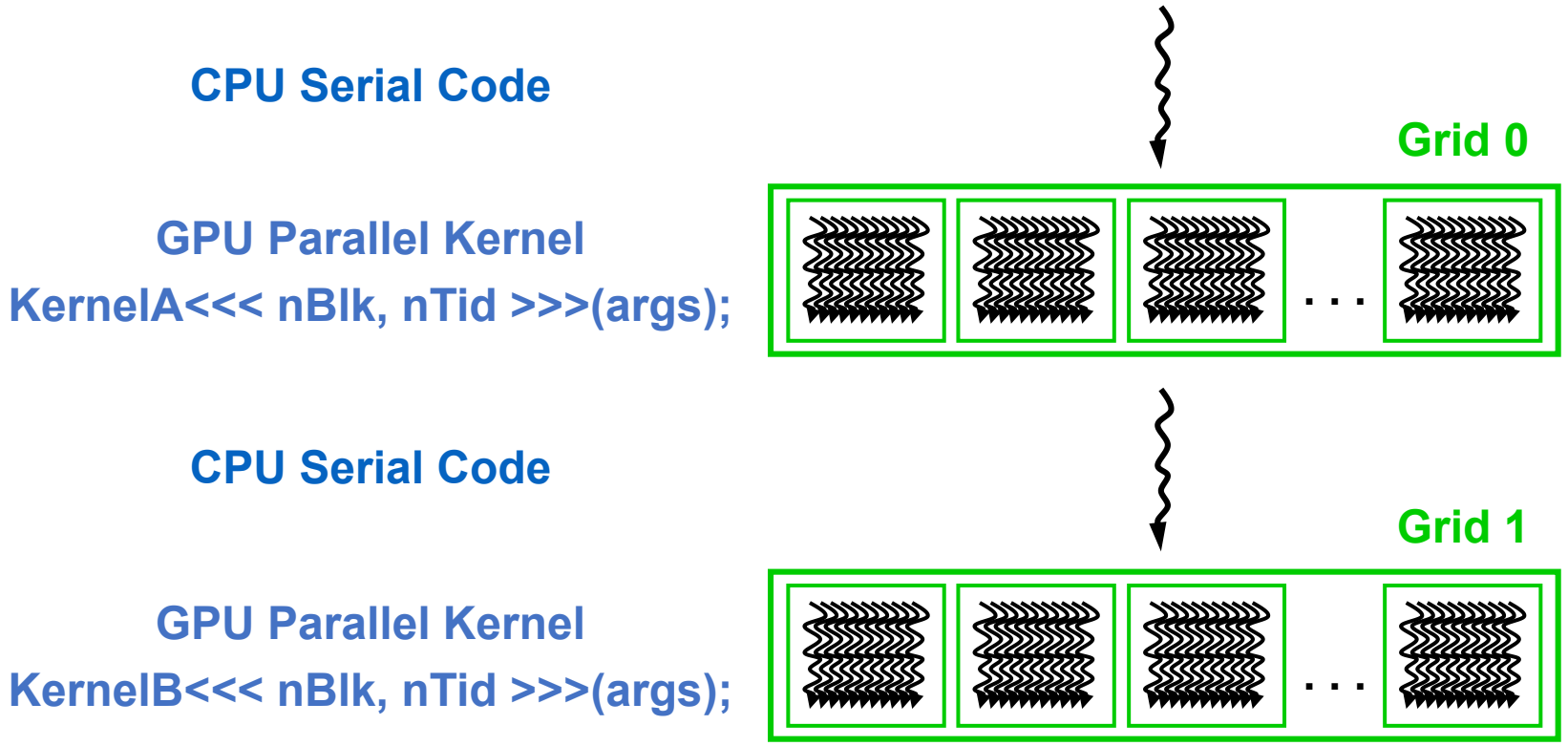
- Task（任务）
- Parallel Task（并行任务）
- Serial Execution（串行执行）
- Parallel Execution （并行执行）
- Shared Memory（共享存储）
- Distributed Memory（分布式存储）
- Communications（通信）
- Synchronization（同步）
- Granularity（粒度）
- Observed Speedup（加速比）
- Parallel Overhead（并行开销）
- Scalability（可扩展性）

并行编程模型

- 共享存储模型 Shared Memory Model
- 线程模型 Threads Model
- 消息传递模型 Message Passing Model
- 数据并行模型 Data Parallel Model

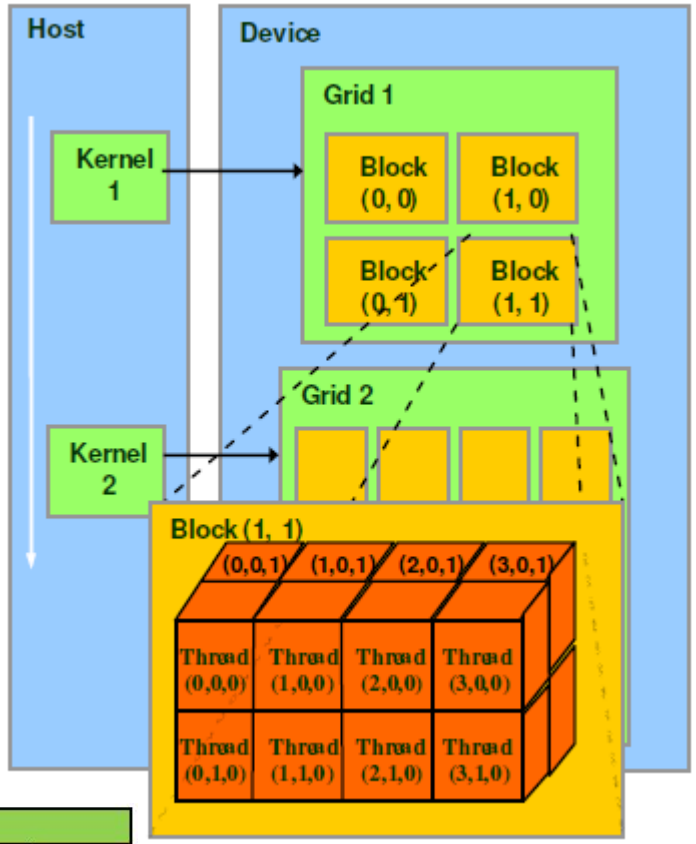
CUDA

- CUDA integrated CPU+GPU application C program
  - Serial or modestly parallel C code executes on CPU
  - Highly parallel SPMD kernel C code executes on GPU

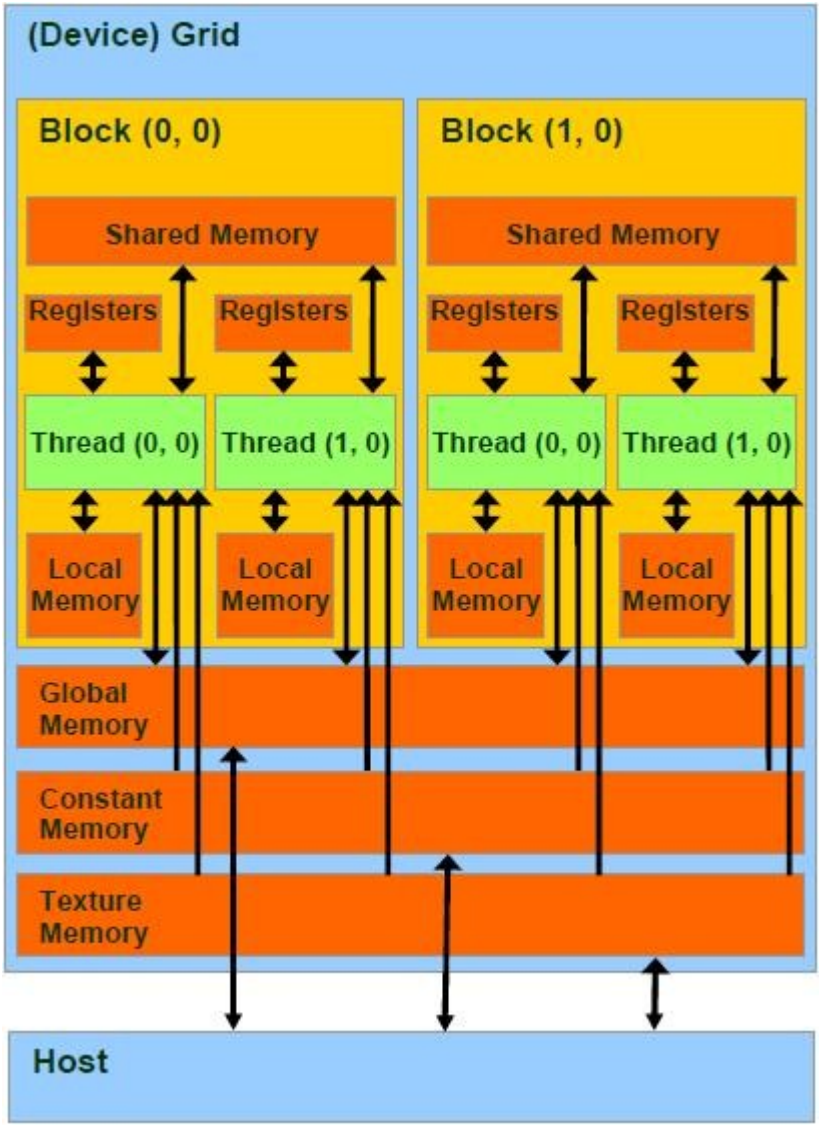


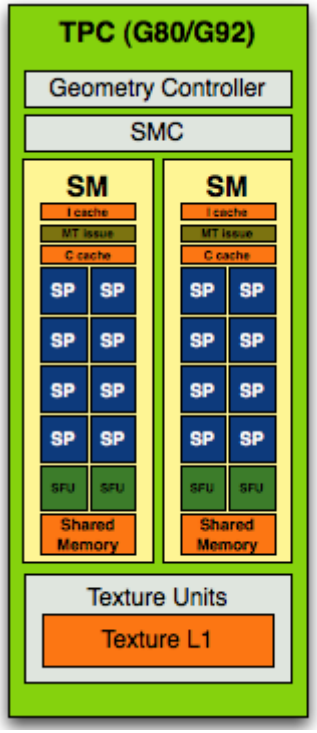
CUDA

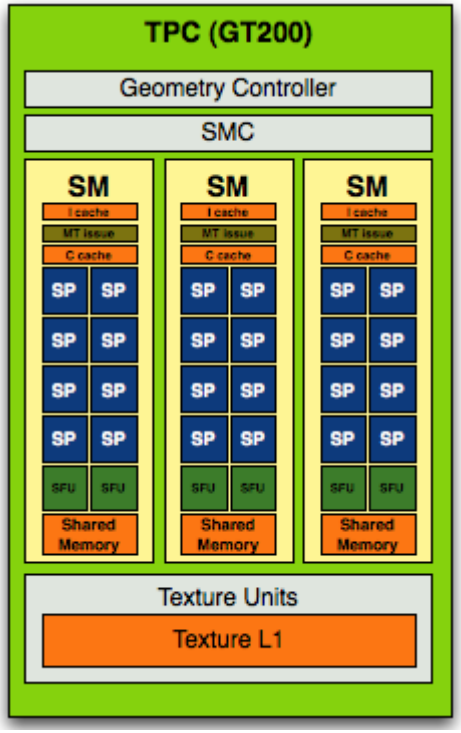
- thread：一个CUDA的并行程序会被以许多个threads来执行。
- block：数个threads会被群组成一个block，同一个block中的threads可以同步，也可以通过shared memory通信。
- grid：多个blocks则会再构成grid。

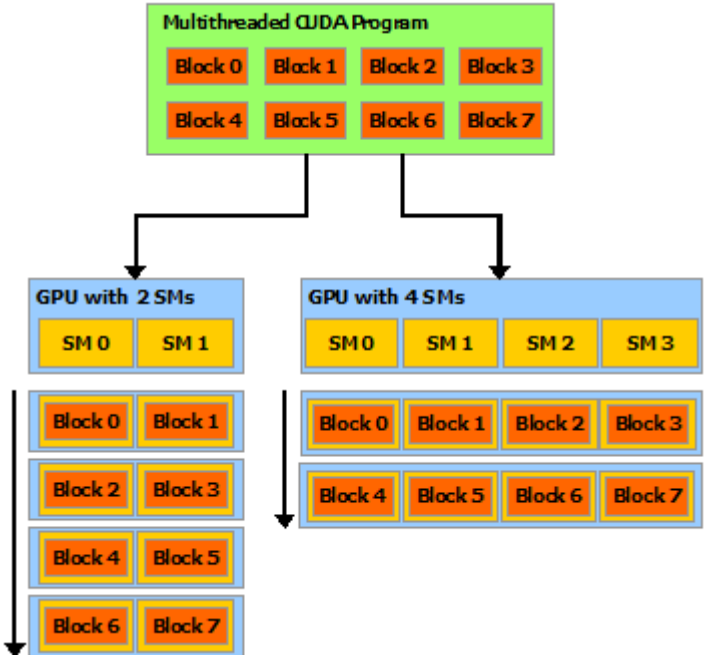


	计算能力							
指标	1	1.1	1.2	1.3	2.x	3	3.5	5
Grid维度	2				3			
Grid的x方向最大Block数	65535					2的31次方减1		
Grid的y/z方向最大Block数	65535							
Block维度	3							
Block的x/y方向最大值	512				1024			
Block的z方向最大值	64							
每个block的最大线程数	512				1024			









Hardware & software

CUDA Devices and Threads

- A compute device
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads in parallel
  - Is typically a GPU but can also be another type of parallel processing device
- Data-parallel portions of an application are expressed as device kernels which run on many threads
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - Very little creation overhead
  - GPU needs 1000s of threads for full efficiency
    - Multi-core CPU needs only a few

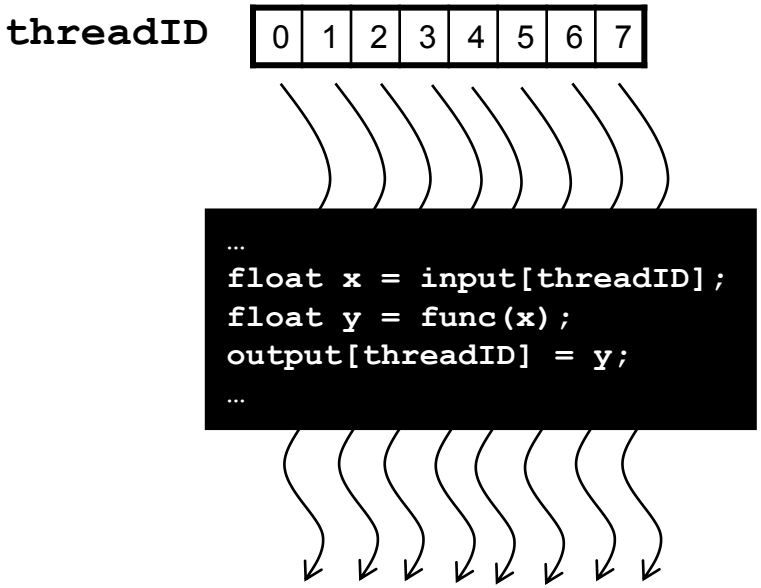
CUDA Extends C

- **Declspecs**
  - **global, device, shared, local, constant**
- **Keywords**
  - **threadIdx, blockIdx**
- **Intrinsics**
  - **__syncthreads**
- **Runtime API**
  - **Memory, symbol, execution management**
- **Function launch**

```
__device__ float filter[N];

__global__ void convolve (float *image)  {

  __shared__ float region[M];
  ...

  region[threadIdx] = image[i];

  __syncthreads()
  ...

  image[j] = result;
}

// Allocate GPU memory
void *myimage = cudaMalloc(bytes)


// 100 blocks, 10 threads per block
convolve<<<100, 10>>> (myimage);
```

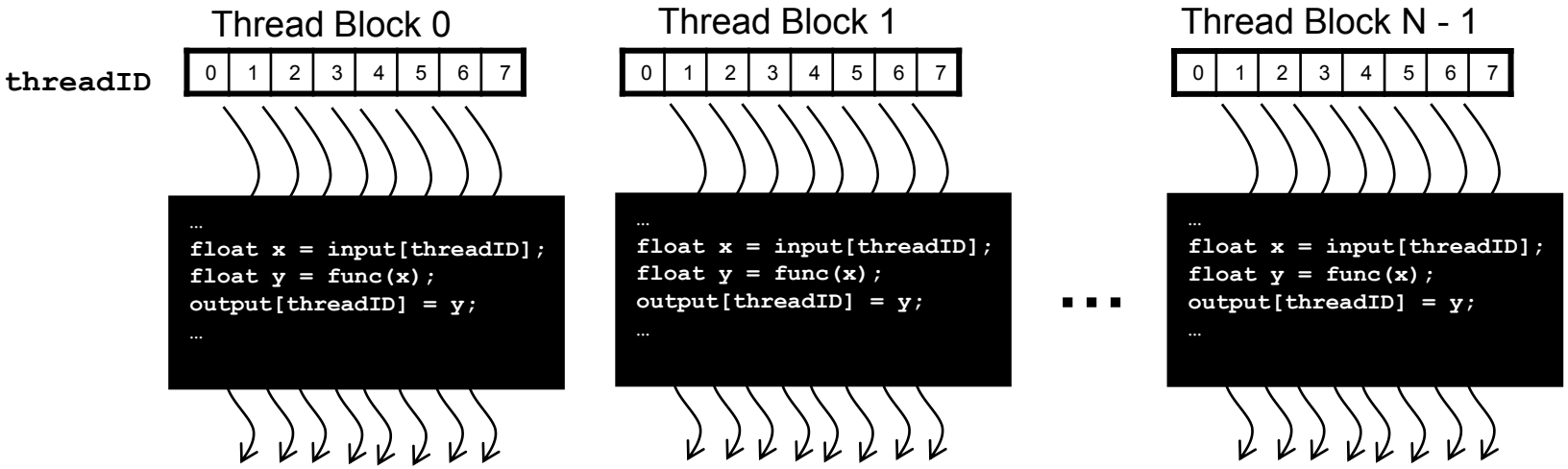
Arrays of Parallel Threads

- A CUDA kernel is executed by an array of threads
  - All threads run the same code (SIMD)
  - Each thread has an ID that it uses to compute memory addresses and make control decisions



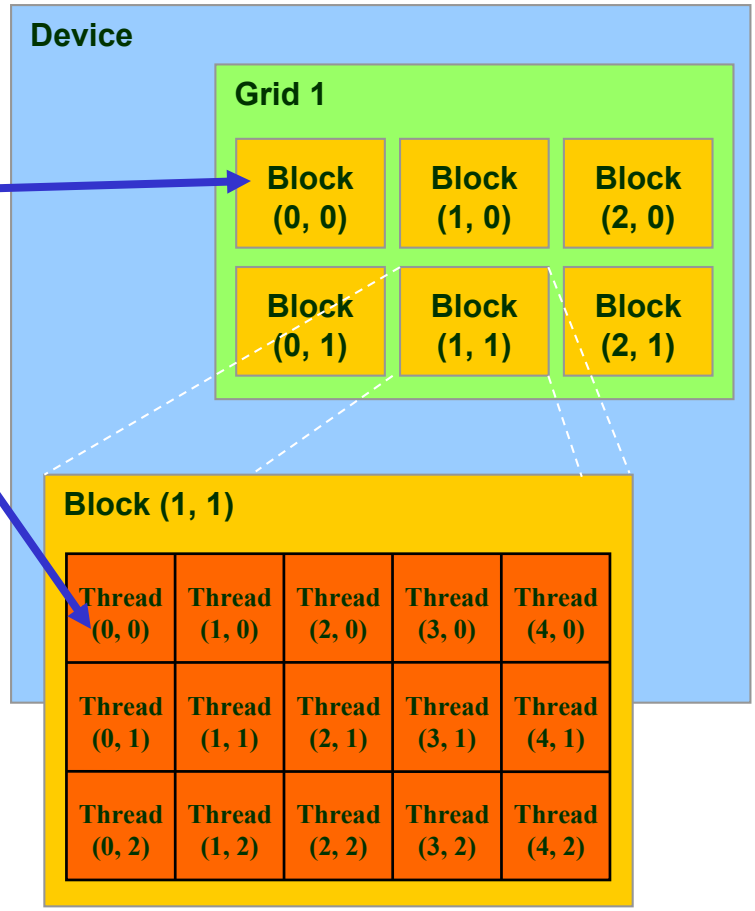
Thread Blocks: Scalable Cooperation

- Divide monolithic thread array into multiple blocks
  - Threads within a block cooperate via **shared memory, atomic operations** and **barrier synchronization**
  - Threads in different blocks cannot cooperate



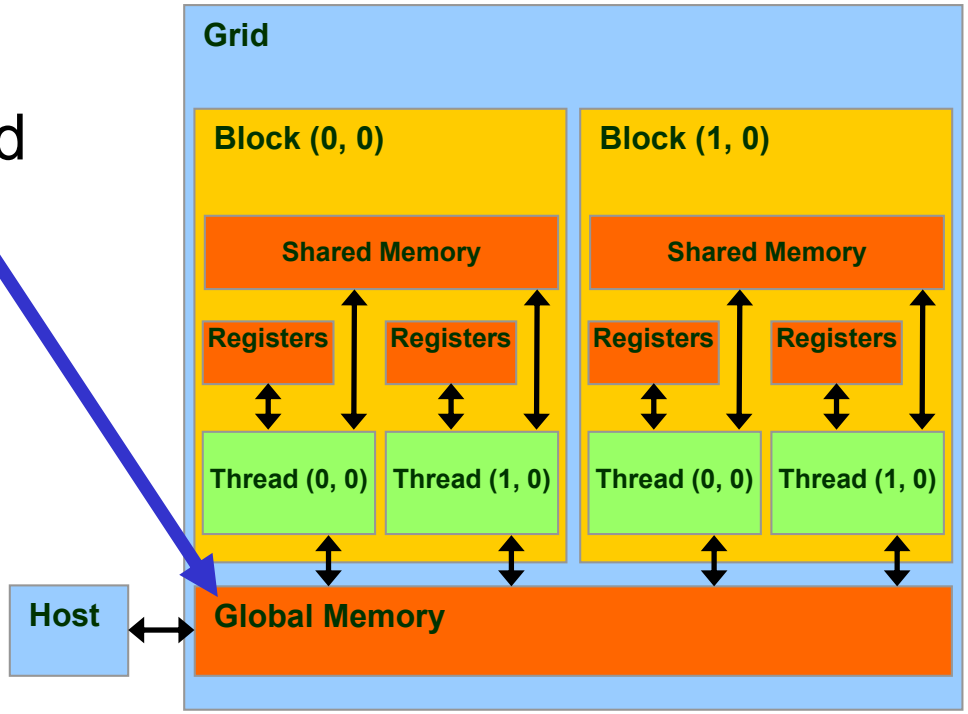
Block IDs and Thread IDs

- Each thread uses IDs to decide what data to work on
  - Block ID: 1D or 2D
  - Thread ID: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes
  - …



CUDA Memory Model Overview

- Global memory
  - Main means of communicating R/W Data between host and device
  - Contents visible to all threads
  - Long latency access
- We will focus on global memory for now



/usr/local/cuda/samples/1_Utilities/deviceQuery

```
Detected 4 CUDA Capable device(s)

Device 0: "TITAN Xp"
  CUDA Driver Version / Runtime Version          8.0 / 8.0
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 12190 MBytes (12782075904 bytes)
  (30) Multiprocessors, (128) CUDA Cores/MP:     3840 CUDA Cores
  GPU Max Clock rate:                            1582 MHz (1.58 GHz)
  Memory Clock rate:                             5705 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 3145728 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  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)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 4 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```

```
top - 20:45:21 up 4 days,  5:13,  7 users,  load average: 4.07, 4.02, 4.00
Tasks: 602 total,   5 running, 597 sleeping,   0 stopped,   0 zombie
%Cpu0  : 14.0 us,  2.1 sy,  0.0 ni, 82.9 id,  0.7 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu2  :  0.0 us,  1.0 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 13.9 us,  2.0 sy,  0.0 ni, 83.4 id,  0.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 57.8 us, 10.9 sy,  0.0 ni, 28.1 id,  2.6 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us, 12.7 sy,  0.0 ni, 87.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 55.0 us, 10.7 sy,  0.0 ni, 28.7 id,  5.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  : 56.0 us, 11.3 sy,  0.0 ni, 32.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 : 82.1 us, 17.9 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.7 us,  0.0 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 : 12.3 us,  2.0 sy,  0.0 ni, 68.8 id, 16.6 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu17 :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.0 us,  2.3 sy,  0.0 ni, 97.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  0.0 us,  2.3 sy,  0.0 ni, 97.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu24 :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu25 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu26 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu27 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu28 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu29 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu30 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu31 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu32 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu33 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu34 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu35 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu36 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu37 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

```
/************************************************************/
// ！！！！！！！！！！！！！注意！！！！！！！！！！！！！！！
/************************************************************/
// grid划分成a维，block划分成b维，
// 等价于
// blocks是a维的，Threads是b维的。
// 这里，本人用的是第一中说法。
/************************************************************/


// 情况1：grid划分成1维，block划分为1维。
__device__ int getGlobalIdx_1D_1D() {
  int threadId = blockIdx.x *blockDim.x + threadIdx.x;
  return threadId;
}

// 情况2：grid划分成1维，block划分为2维。
__device__ int getGlobalIdx_1D_2D() {
  int threadId = blockIdx.x * blockDim.x * blockDim.y
    + threadIdx.y * blockDim.x + threadIdx.x;
  return threadId;
}

// 情况3：grid划分成1维，block划分为3维。
__device__ int getGlobalIdx_1D_3D() {
  int threadId = blockIdx.x * blockDim.x * blockDim.y * blockDim.z
    + threadIdx.z * blockDim.y * blockDim.x
    + threadIdx.y * blockDim.x + threadIdx.x;
  return threadId;
}

// 情况4：grid划分成2维，block划分为1维。
__device__ int getGlobalIdx_2D_1D() {
  int blockId = blockIdx.y * gridDim.x + blockIdx.x;
  int threadId = blockId * blockDim.x + threadIdx.x;
  return threadId;
}

// 情况5：grid划分成2维，block划分为2维。
__device__ int getGlobalIdx_2D_2D() {
  int blockId = blockIdx.x + blockIdx.y * gridDim.x;
  int threadId = blockId * (blockDim.x * blockDim.y)
    + (threadIdx.y * blockDim.x) + threadIdx.x;
  return threadId;
}

// 情况6：grid划分成2维，block划分为3维。
__device__ int getGlobalIdx_2D_3D() {
  int blockId = blockIdx.x + blockIdx.y * gridDim.x;
  int threadId = blockId * (blockDim.x * blockDim.y * blockDim.z)
    + (threadIdx.z * (blockDim.x * blockDim.y))
    + (threadIdx.y * blockDim.x) + threadIdx.x;
  return threadId;
}

// 情况7：grid划分成3维，block划分为1维。
__device__ int getGlobalIdx_3D_1D() {
  int blockId = blockIdx.x + blockIdx.y * gridDim.x
    + gridDim.x * gridDim.y * blockIdx.z;
  int threadId = blockId * blockDim.x + threadIdx.x;
  return threadId;
}

// 情况8：grid划分成3维，block划分为2维。
__device__ int getGlobalIdx_3D_2D() {
  int blockId = blockIdx.x + blockIdx.y * gridDim.x
    + gridDim.x * gridDim.y * blockIdx.z;
  int threadId = blockId * (blockDim.x * blockDim.y)
    + (threadIdx.y * blockDim.x) + threadIdx.x;
  return threadId;
}

// 情况9：grid划分成3维，block划分为3维。
__device__ int getGlobalIdx_3D_3D() {
  int blockId = blockIdx.x + blockIdx.y * gridDim.x
    + gridDim.x * gridDim.y * blockIdx.z;
  int threadId = blockId * (blockDim.x * blockDim.y * blockDim.z)
    + (threadIdx.z * (blockDim.x * blockDim.y))
    + (threadIdx.y * blockDim.x) + threadIdx.x;
  return threadId;
}
```