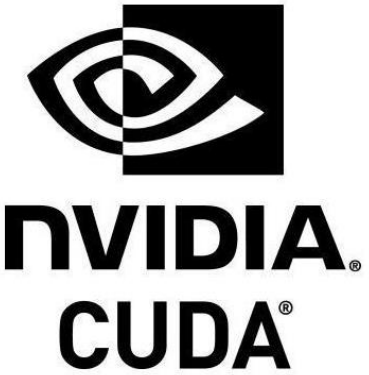
CS3210

Parallel Computing

Changes from Monday in Green



Lab 3

Mon (4pm)
Tues (2pm)

Admin
Updates

- Assignment 1 Part 2 was due 11am yesterday
  - Late penalty: 10% per day, up to a week
  - Please re-download your submission to check that it contains: your programs, testcases, scripts and report

- Midterms: Week 8 during lecture
  - 15% (open-book)
  - Syllabus: Lectures 1 - 6, Tutorials 1 - 2, Labs 1 - 3
  - Wait for LumiNUS announcement for more details (not Zoom proctored)
  - Tutorial and lab recordings: check LumiNUS

Admin
Compute Cluster Machine Info

- Access Compute Cluster via Sunfire: hostnames below
  - Reserved from: 27 Sep - 5 Nov (accessible 24/7)

Node	xgpc5 - xgpc9	xgpf5 - xgpf9
OS	Ubuntu 16.04.6 LTS	Ubuntu 18.04.2 LTS
CPU	Dual-socket Xeon Silver 4108 (8 p. cores with SMT)	Dual-socket Xeon Silver 4116 (12 p. cores with SMT)
RAM	128 GB	256 GB
GPU	Tesla V100 [CC = 7.0] (32GB HBM2, 80 SMs, 5120 CUDA cores)	Tesla T4 [CC = 7.5] (16GB GDDR6, 40 SMs, 2560 CUDA cores)

Admin
Roadmap

- No lab submission this week ☺
  - Lab exercises are mostly self-exploratory

- Today's lab: general-purpose GPU programming (GPGPU)
  - Platform: Compute Unified Device Architecture (CUDA) for Nvidia GPUs
  - Part 1: CUDA functions, threads and kernels
  - Part 2: CUDA memory model
  - Part 3: CUDA synchronization primitives

Part 1

nvidia-smi Utility

- Reports details and utilisation statistics of Nvidia GPUs
  - Command: **nvidia-smi** (use with **watch** to run repeatedly)

```
keven@xgpe2:~$ nvidia-smi
Mon Sep 28 14:10:24 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.67       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  TITAN RTX           On   | 00000000:D8:00.0 Off |                  N/A |
| 41%   46C    P8    34W / 280W |   2554MiB / 24190MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     30461      C   ...09838/anaconda3/envs/gcn10.1/bin/python  2543MiB |
+-----------------------------------------------------------------------------+
```

Part 1

nvprof Utility

- Profiles execution of CUDA programs, similar to **perf**
  - Command: **nvprof <executable>**

```
keven@xgpc5:~/L3$ nvprof ./a.out
==435786== NVPROF is profiling process 435786, command: ./a.out
Last CUDA error no kernel image is available for execution on the device
==435786== Profiling application: ./a.out
==435786== Profiling result:
No kernels were profiled.
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
     API calls:   99.73%  233.01ms         1  233.01ms  233.01ms  233.01ms  cudaLaunchKernel
                   0.17%  393.54us         1  393.54us  393.54us  393.54us  cuDeviceTotalMem
                   0.07%  160.55us       101  1.5890us     150ns  69.340us  cuDeviceGetAttribute
                   0.01%  31.035us         1  31.035us  31.035us  31.035us  cuDeviceGetName
                   0.01%  21.312us         1  21.312us  21.312us  21.312us  cuDeviceGetPCIBusId
                   0.00%  6.4480us         2  3.2240us     157ns  6.2910us  cuDeviceGet
                   0.00%  4.4720us         1  4.4720us  4.4720us  4.4720us  cudaDeviceSynchronize
                   0.00%  1.5780us         3     526ns     174ns  1.0750us  cuDeviceGetCount
                   0.00%  1.0790us         1  1.0790us  1.0790us  1.0790us  cudaGetErrorString
                   0.00%     376ns         1     376ns     376ns     376ns  cudaGetLastError
                   0.00%     311ns         1     311ns     311ns     311ns  cuDeviceGetUuid
```

Part 1

CUDA functions

- CUDA programs are modified C/C++ programs with sections of CUDA code
  - Terminology: host - CPU, device - GPU
  - CUDA code executes on the GPU, and are known as kernels

- Need to explicitly mark functions running on host/device with additional modifier keywords
  - **`__host__`** - default; can only be invoked on the host
  - **`__global__`** - invoked from host/device to run on device
  - **`__device__`** - invoked only from device to run on device

Part 1

CUDA kernel invocation

- Invoking a CUDA kernel function (those annotated with `__global__`) launches a new grid of thread blocks
  - Grid: 1-, 2- or 3- dimensional array
  - Each block is itself a grid of CUDA threads
  - All blocks in grid have same thread layout

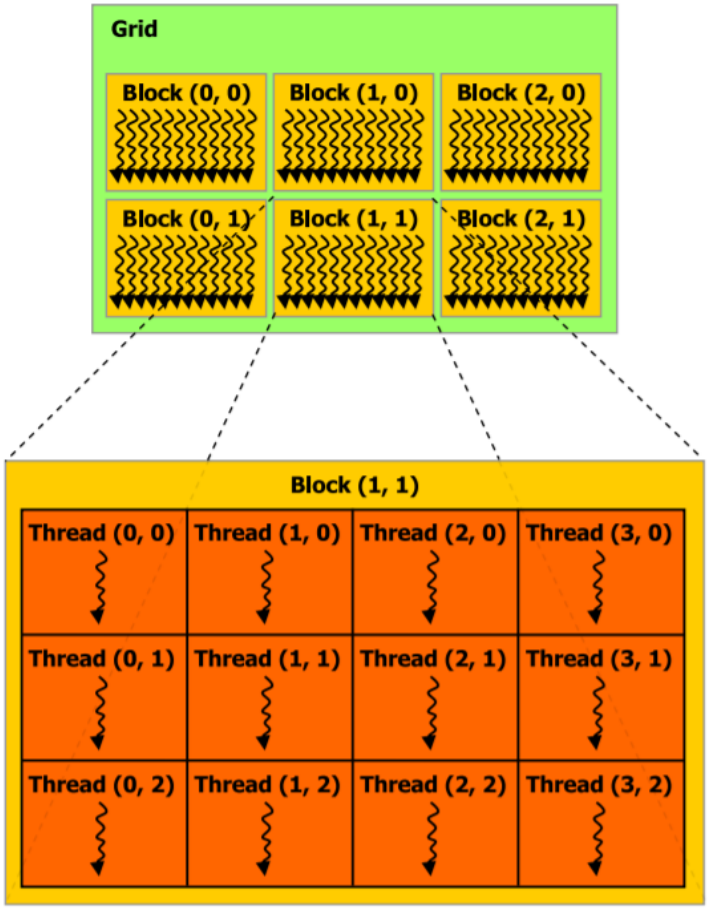
- Specify block and grid dimensions when invoking kernel:
  `kernel_name<<<griddim, blockdim>>>(<args>)`
  - For 1D, specify an integer
  - Otherwise, declare a variable of type **`dim3`** with the sizes as **`<var_name>(<x-size>, <y-size>, <z-size>)`**

Part 1

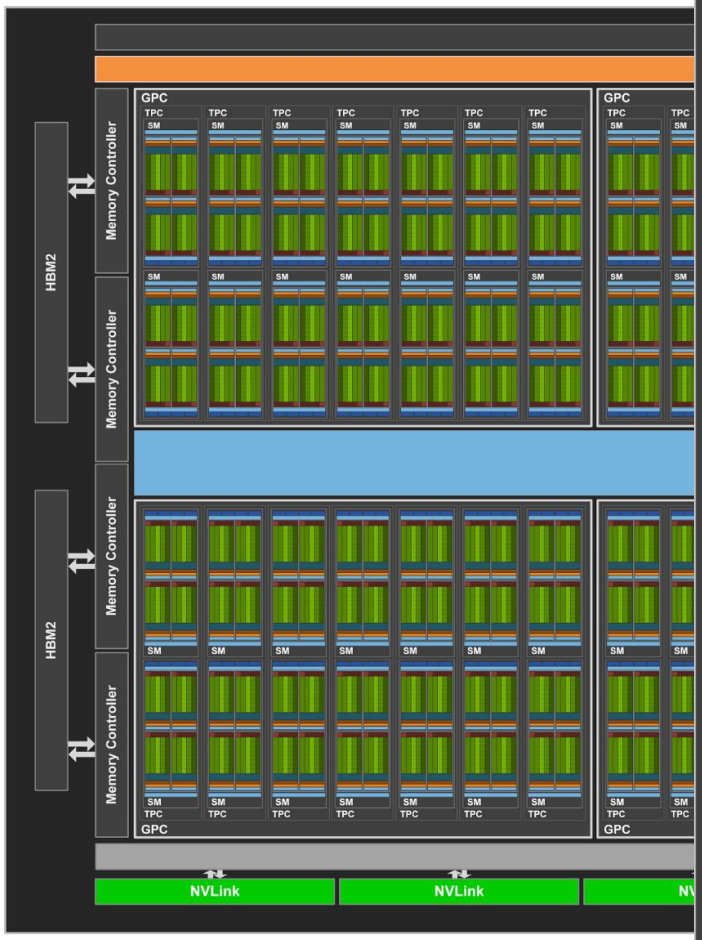
CUDA grid and block layout

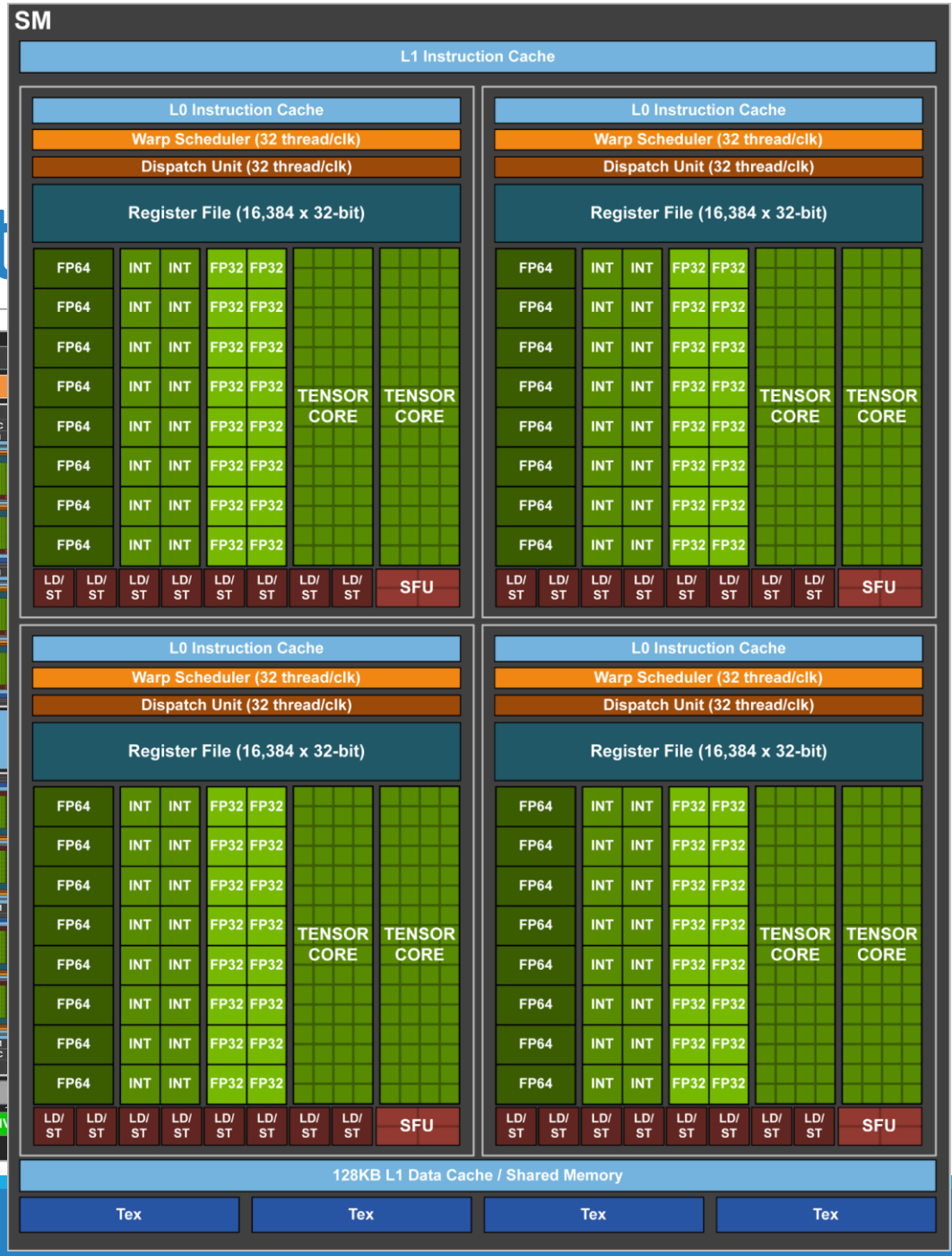
- How do we launch a kernel with this layout?
  - 2D grid with dims (3, 2)
  - 2D block with dims (4, 3)

- Can retrieve
  - Grid and block dims: **`gridDim`** and **`blockDim`**
  - Block and thread indexes: **`blockIdx.<dim>`** and **`threadIdx.<dim>`**



Part 1
GPU Architect





Part 1

Block and warp scheduling

- GPU comprised of Streaming Multiprocessors (SMs)
  - When a kernel is executed, runtime assigns blocks to SMs; once assigned, block does not migrate
  - Block divided into *warps*, collections of (up to*) 32 threads
  - Execution context of all warps on an SM are stored at all times → essentially free context-switches
  - Threads from different blocks cannot cooperate

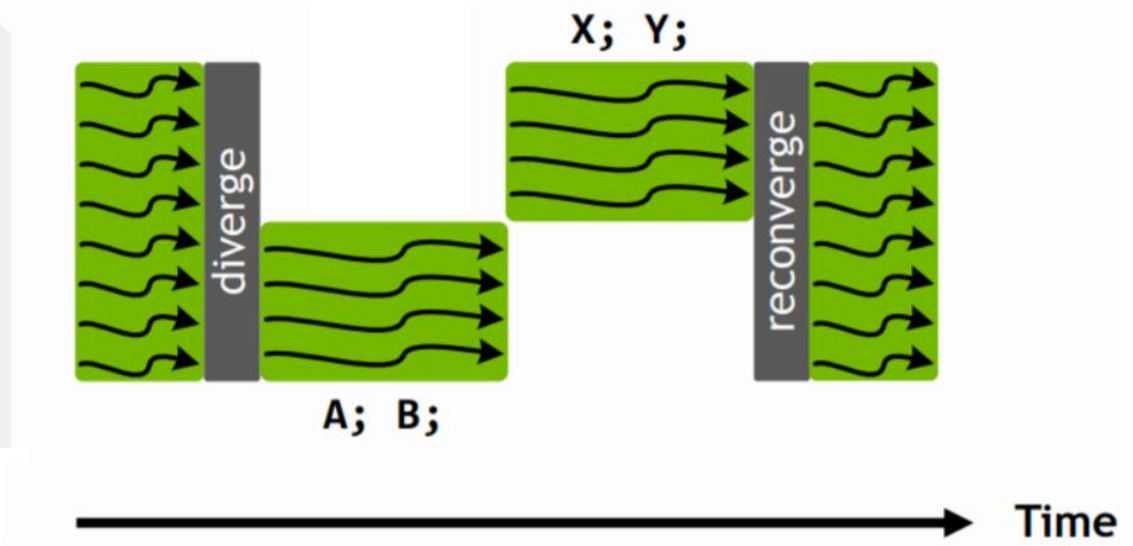
- Each cycle: each SM scheduler picks one ready warp and issues a single instruction from it (SIMT)
  - Each SM can execute several warps in parallel on diff. EUs

Part 1

Warp divergence

- How are branches in CUDA code handled?

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
```

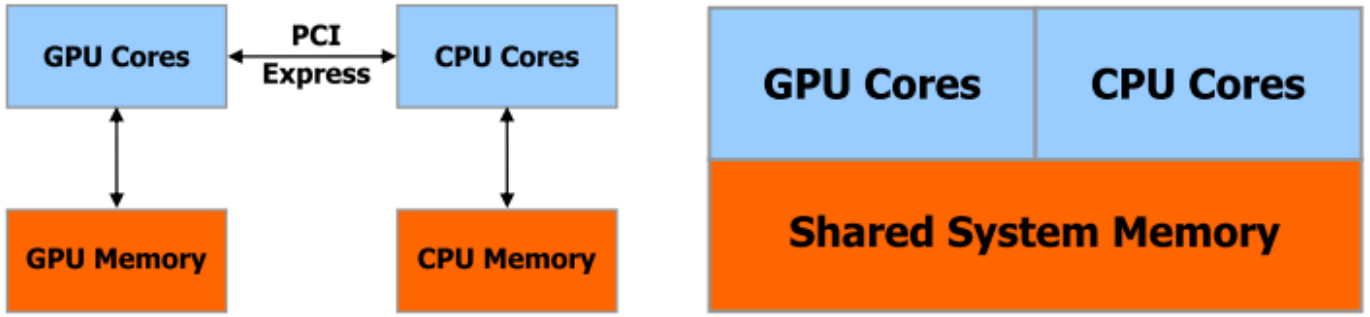


- Warp divergence (from branches or intra-warp synchronization) results in significant performance loss!

Part 2

GPU memory model

- Two types of GPU: integrated and discrete

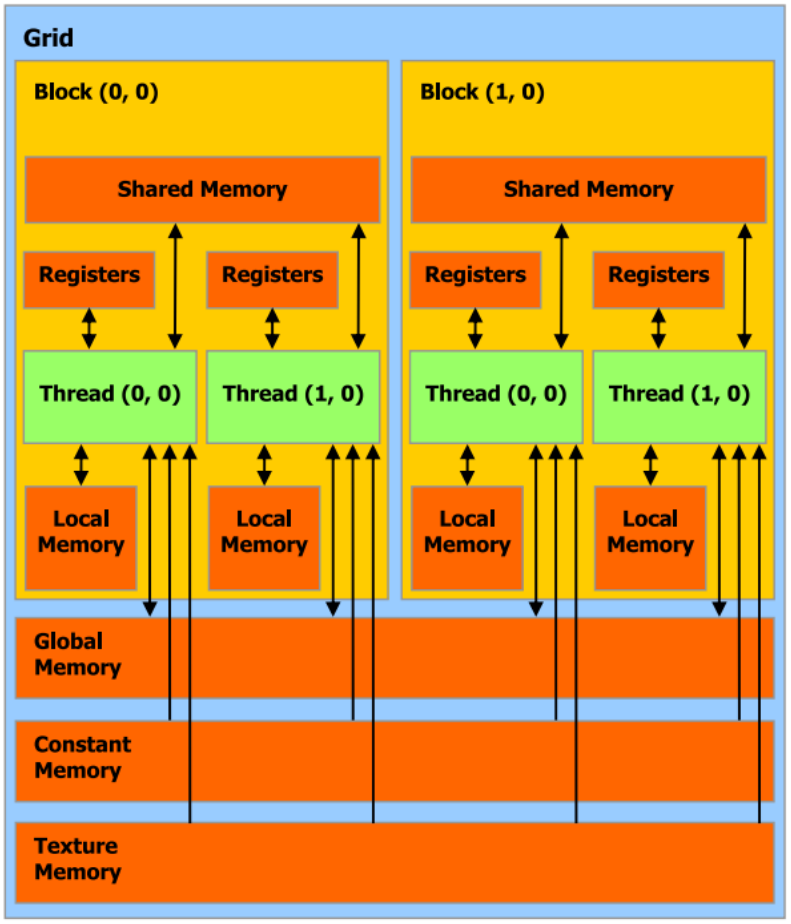


- For CUDA with Nvidia GPUs, the memory types available:
  - Per-thread: registers (implicitly allocated), local* variables
  - Per-block (explicit sync): shared memory
  - Per-program (explicit sync): global, constant, texture

Part 2

CUDA memory hierarchy

- Latency (fastest to slowest)
  - Registers < shared < texture < local ≲ global
- Notable details
  - 128 KB on each SM configured between L1D$/shared memory
  - **Local memory: in fact closer to "thread-private" global memory**
  - Compiler decides whether to use registers or local memory



Part 2

CUDA memory allocation

- Kernel invocation arguments are passed to device using constant memory (limited to 4KB)
- Declaration syntax for different memory types can only be used with fixed size types, i.e. primitives or `structs`

- To allocate memory dynamically from the GPU memory, use `cudaMalloc` (this declares a linear array)
- Statically or dynamically allocated memory on the device cannot be accessed by host directly
  - Either copy manually (**`cudaMemcpy`**), or use managed (unified) memory (**`__managed__`**/**`cudaMallocManaged`**)

Part 3

CUDA synchronisation

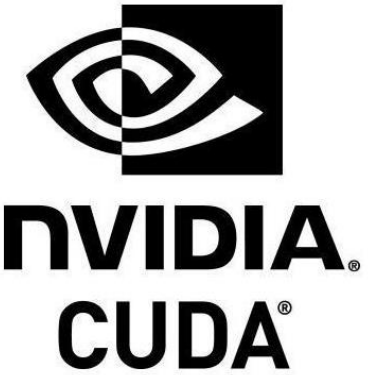
- CUDA provides a set of atomic functions for updating a 32-bit or 64-bit shared memory location
  - This shared memory can be intra-block or global
  - Allows us to enforce mutual exclusion in three ways:
  - Across program on both host/device, **`atomicAdd_system`**
  - Across threads from same program on device: **`atomicAdd`**
  - Across threads **in the same block**: **`atomicAdd_block`**

- We can also synchronise all threads (barrier) in a block with **`__syncthreads()`**

CS3210

Parallel Computing

Thank you! Any questions?



Lab 3

Mon (4pm)
Tues (2pm)

bit.ly/cs3210-t01-qn