LAU
الجامعة اللبنانية الأميركية
Lebanese American University

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

Introduction CUDA C

Instructor: Haidar M. Harmanani

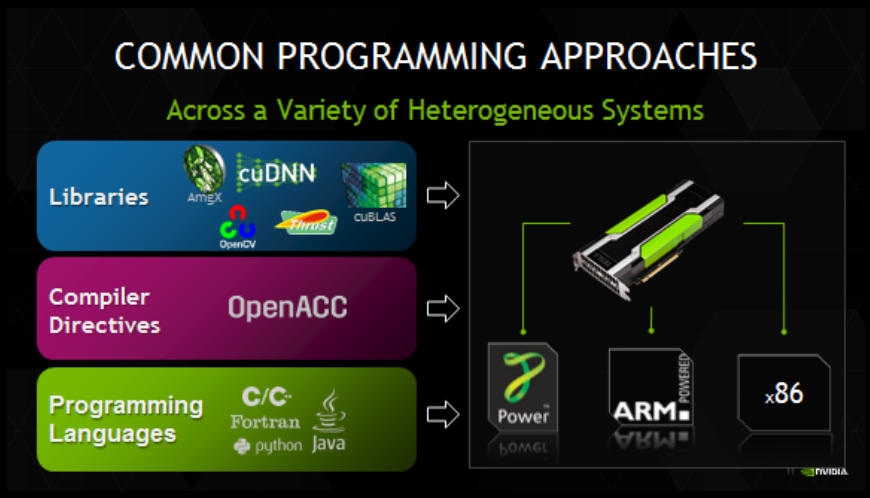
Spring 2017

## CONCEPTS

- Heterogeneous Computing
- Hello World!
- Compiling

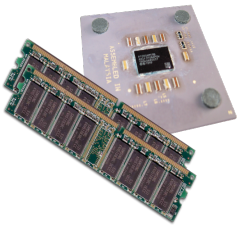
- CUDA is a parallel computing platform and programming model invented by NVIDIA
- Enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU)





# Heterogeneous Computing

- Terminology:
  - *Host* The CPU and its memory (host memory)
  - *Device* The GPU and its memory (device memory)

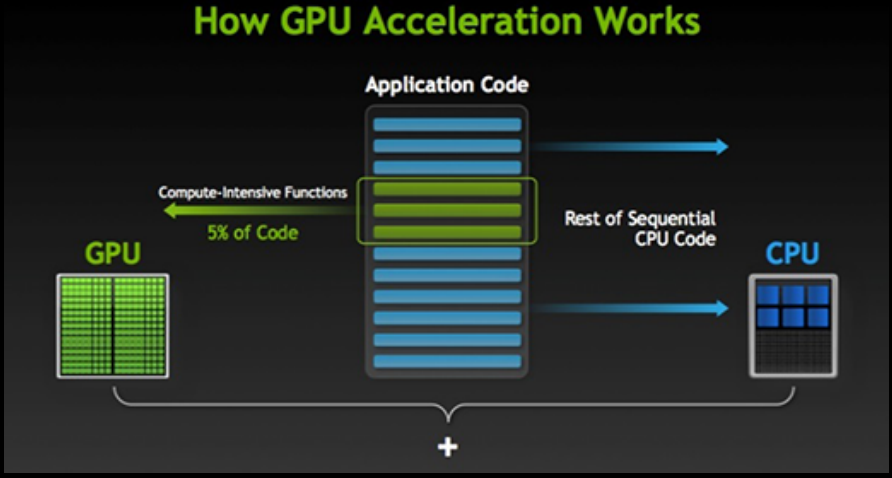


Host

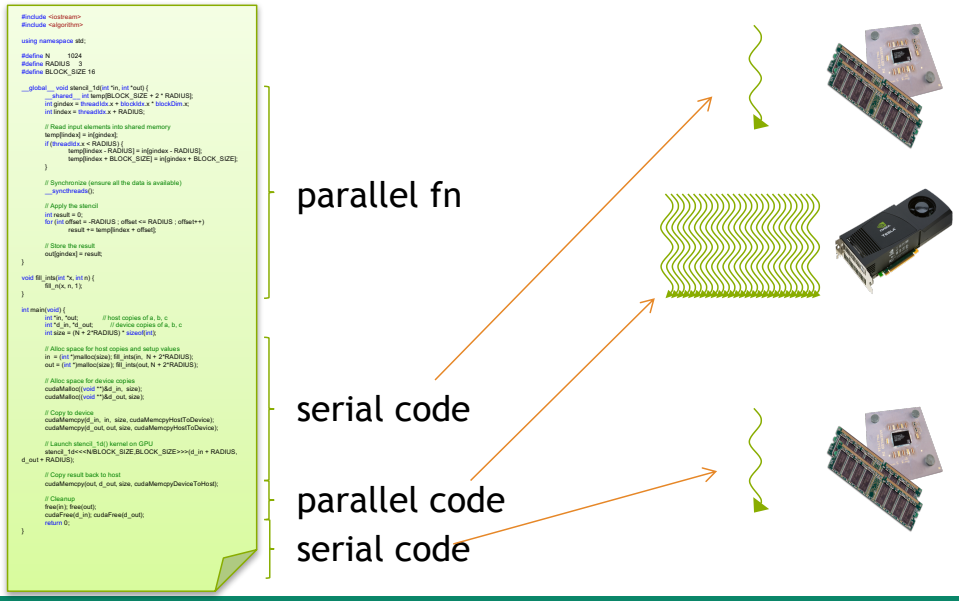
Device



# Heterogeneous Parallel Computing

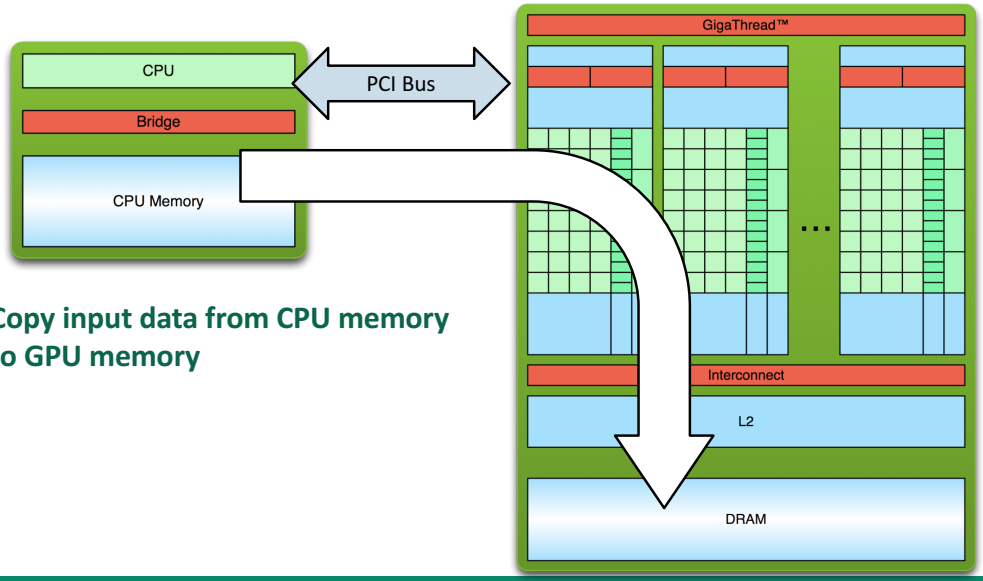


# Heterogeneous Computing



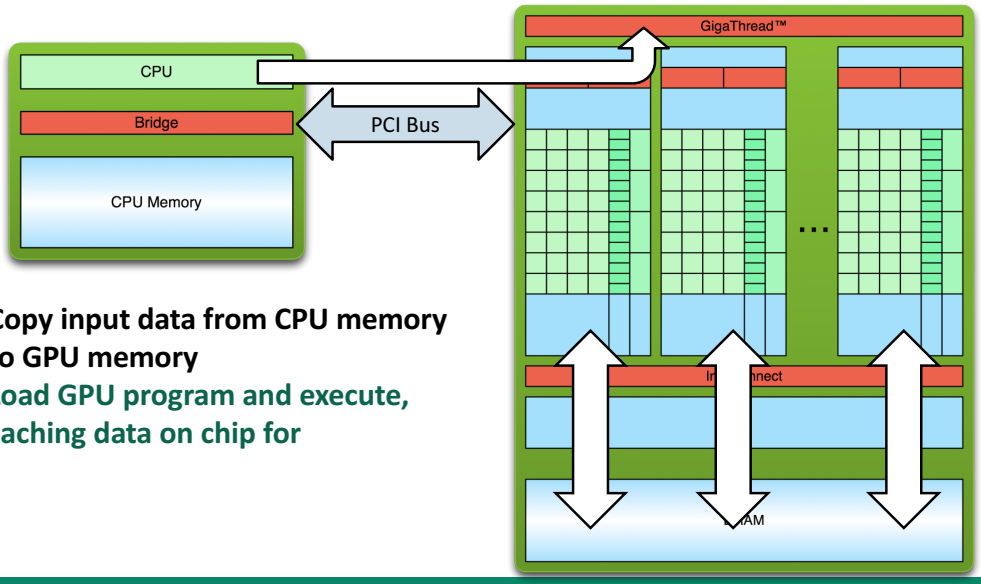



# Simple Processing Flow




1. **Copy input data from CPU memory to GPU memory**

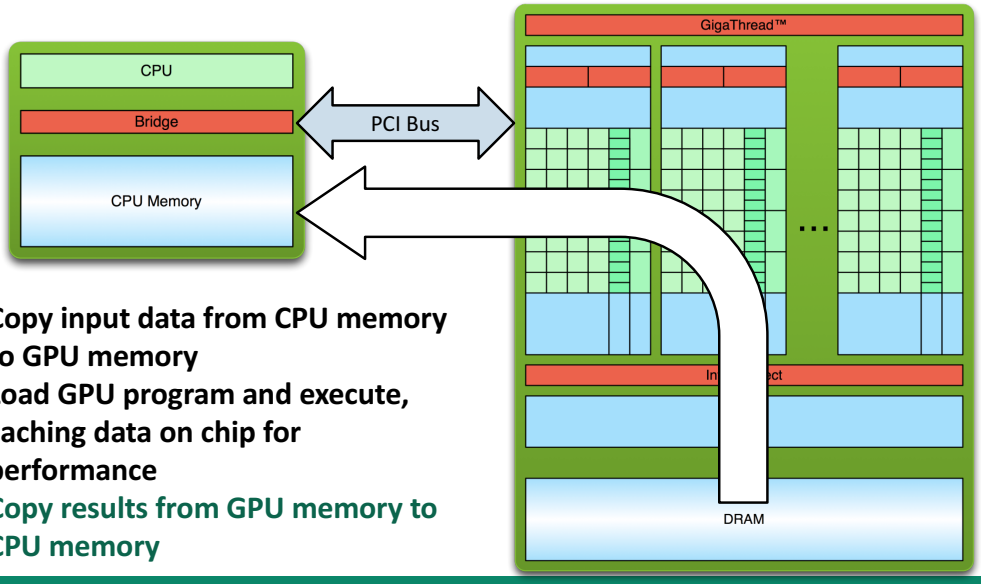
# Simple Processing Flow



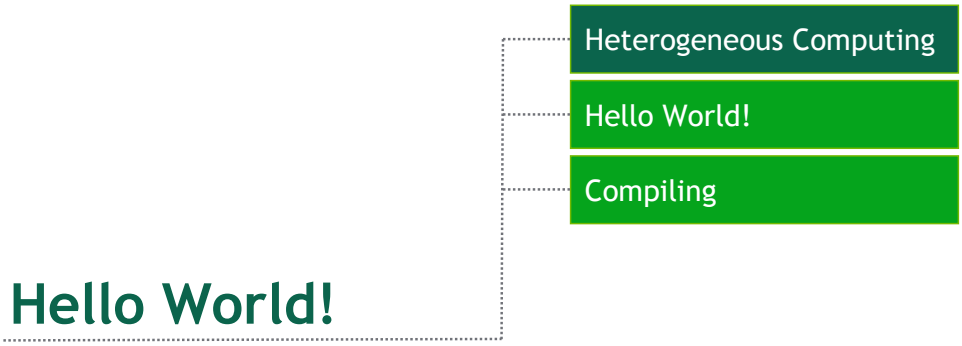

1. **Copy input data from CPU memory to GPU memory**
2. **Load GPU program and execute, caching data on chip for**



# Simple Processing Flow




1. **Copy input data from CPU memory to GPU memory**
2. **Load GPU program and execute, caching data on chip for performance**
3. **Copy results from GPU memory to CPU memory**




# CUDA

- CUDA provides a simple grid model
  - Can decompose into CUDA blocks
- A linear distribution of work within a single block leads to an ideal decomposition into CUDA blocks.
  - Can assign up to sixteen blocks per SM and we can have up to 16 SMs (32 on some GPUs), any number of blocks of 256 or larger is fine.
  - In practice, limit the number of elements within the block to 128, 256, or 512

# Blocks, Threads, and Grids

- CUDA C allows the definition of a group of blocks in two dimensions
- Use two-dimensional indexing for problems such as matrix math or image processing
  - Avoid annoying translations from linear to rectangular indices
- A collection of parallel blocks is called a grid
- No dimension of a launch of blocks may exceed 65,535



# CUDA Hello World

- We will discuss over the following few slides a couple of simple examples
- We will start with the classical `hello_world` example

# Hello World (Host Only)

- Recall
  - *Host*–The CPU and its memory (host memory)
  - *Device*–The GPU and its memory (device memory)

```
int main( void) {
        printf( "Hello, World!\n" );
        return 0;
}
```



# Hello World (Device Code)

- Recall
  - *Host*–The CPU and its memory (host memory)
  - *Device*–The GPU and its memory (device memory)

```
__global__  void kernel( void) {
}
int main( void) {
    kernel<<< 1, 1 >>>();
    printf( "Hello, World!\n" );
    return 0;
}
```

**Output:**

```
$ nvcc hello.cu
$ a.out
Hello World!
$
```

# Hello World (Device Code)

- CUDA C keyword `__global__` indicates that a function
  - Runs on the device
  - Called from host code

```
kernel<<< 1, 1 >>>();
```

  - Triple angle brackets mark a call from hostcode to devicecode
  - Sometimes called a "kernel launch"
- nvcc splits the source file into host and device components
  - NVIDIA's compiler handles device functions like kernel()
  - Standard host compiler handles host functions like main()
    - gcc
    - Microsoft Visual C