EBU7501: Cloud Computing
Week 2, Day 1: Introduction to GPU

Dr. Gokop Goteng

Queen Mary
University of London

Lecture Aim and Outcome

- Aim
  - The aim of this lecture is to introduce students to graphics processing units (GPUs) architecture
- Outcome
  - At the end of this lecture students should be able to:
    - Identify and explain the main components of GPU architecture
    - Know the different methods of GPU programming
    - Know NVIDIA's G80 GPU architecture

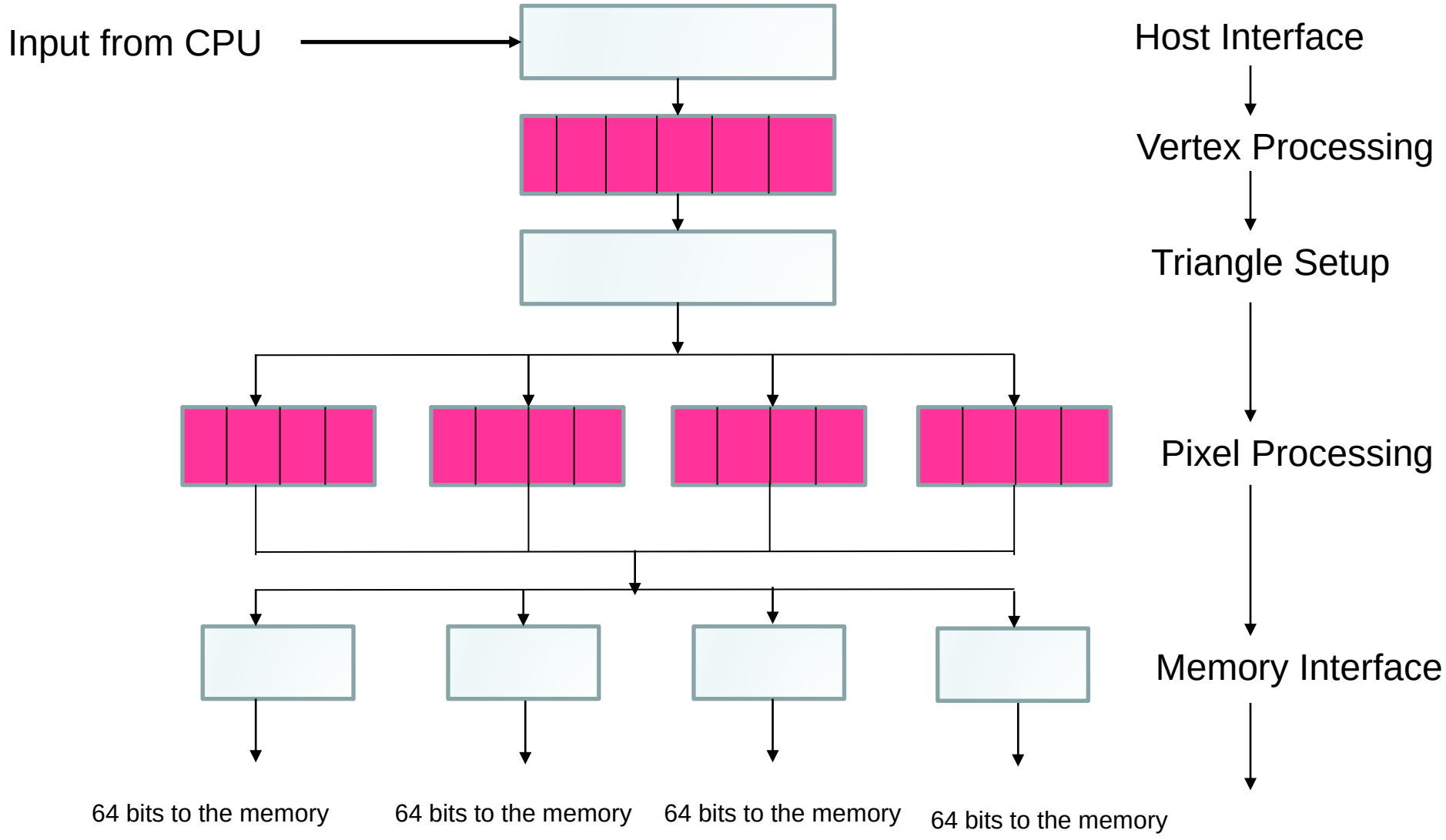
Introduction to GPU

- GPU stands for Graphics Processing Unit
  - GPU is a special electronic circuit which has capability to rapidly manipulate and change computer memory so as to accelerate computation and processing of images and data for output to a display device
- The current programmable GPU was invented by NVIDIA in 1999
  - Though GPUs were around since the early 1990s
- GPUs are also known as Visual Processing Units (VPUs)
- It was initially used mainly by the computer gaming industry for artists and game programming
  - GPUs are very efficient in computer graphics and image processing
- Researchers also started using GPUs to speed up computations using GPUs high performance floating points and parallel capabilities

Introduction to GPU

- GPU is a many-core processor used in parallel high performance computation and processing to accelerate (speed up) the computations in computer systems
- The General Purpose GPU (GPGPU) was also introduced by NVIDIA to allow high level programming languages such as C, C++ to be used to program GPUs
- The Compute Unified Device Architecture (CUDA) is parallel computing programming model invented by NVIDIA for programming GPU and GPGPU

Architecture of GPU



Host Interface

- Host interface provides the communication link between the CPU and GPU
- It produces outputs of streams of vertices which include information on the vertices coordinates, texture, colour, e.t.c.
- It receives commands from the CPU and also obtains information about the geometry from the memory

Vertex Processing

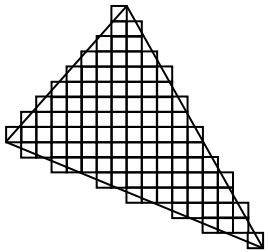
- The vertex processing gets vertices from the host interface in object space and then used them to produce outputs in screen space
- This is usually either done by linear transformation or a complex operation
- No new vertices are created in the operation
- No vertices are also discarded
  - The number of inputs is equal to the number of outputs
  - This is a 1:1 mapping relationship

Triangle Setup

- The triangle setup converts the geometry information into the “raster” format
  - This means that the screen space geometry is the input and the pixels are the output
- Triangles that are located outside the viewing scope are rejected

Triangle Setup

- The triangle setup converts the geometry information into the “raster” format
  - This means that the screen space geometry is the input and the pixels are the output
- Triangles that are located outside the viewing scope are rejected
- Fragments are generated only if their centres are located within the triangle



Pixel or Fragment Processing

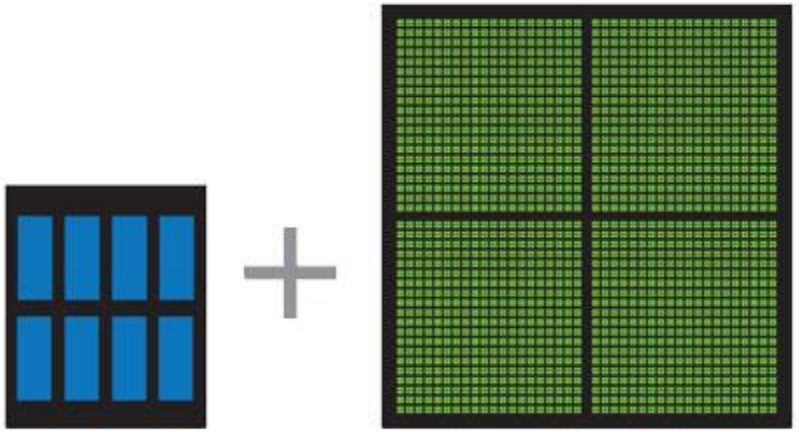
- Each fragment provided by triangle setup is fed into pixel processing as a set of attributes (position, normal, coordinates, etc), which are used to compute the final color for this pixel
- This process consists of texture mapping and math operations
- This is very computationally intensive and is one of the challenges in GPU operations

Memory Interface

- The fragment colors provided by the pixel/fragment processing are stored in the memory buffer
- Colors are then compressed to reduce buffer bandwidth
- It is the second most challenging aspect of GPU operations

The Differences between GPUs and CPUs

- GPUs and CPUs are architecturally very different devices.
  - CPUs are designed for running a small number of potentially quite complex tasks
    - Aims at running systems that execute a number of individually speate and distinct unconnected tasks e.g. operating systems and application software
  - GPUs are designed for running a large number of quite simple tasks
    - Aims at running problems that can be broken into thousands of tiny parts and worked on individually
- CPUs consist of few number of cores (processors) designed primarily for serial sequential (one step at a time) processing
- GPUs consist of massively parallel architecture with thousands of efficient cores that can handle multiple tasks at a time simultaneously



Source: NVIDIA

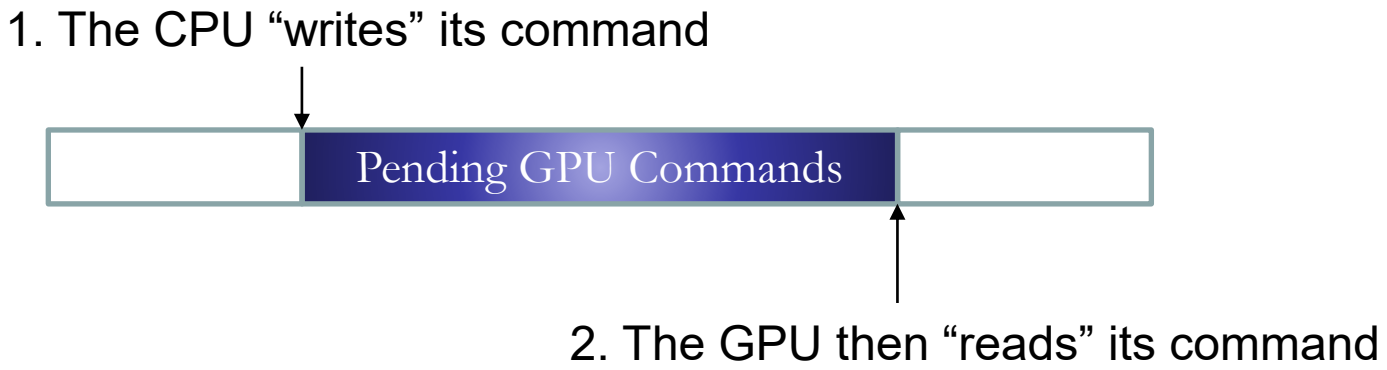
The Differences between GPUs and CPUs

- GPUs have very much faster memory interfaces than CPUs
- GPUs have many parallel execution units and transistor counts while CPUs have fewer execution units and clock speed
- GPUs have more pipelines than CPUs
  - GPUs receive geometry inputs from CPUs and generate pictures as outputs

GPU Pipelines

GPU and CPU Communications

- GPU and CPU that resides in the PC interact in parallel with each other
- There are two separate “threads” that are created with one for the GPU the other for the CPU
  - These threads allow the GPU and CPU to communicate through a command buffer



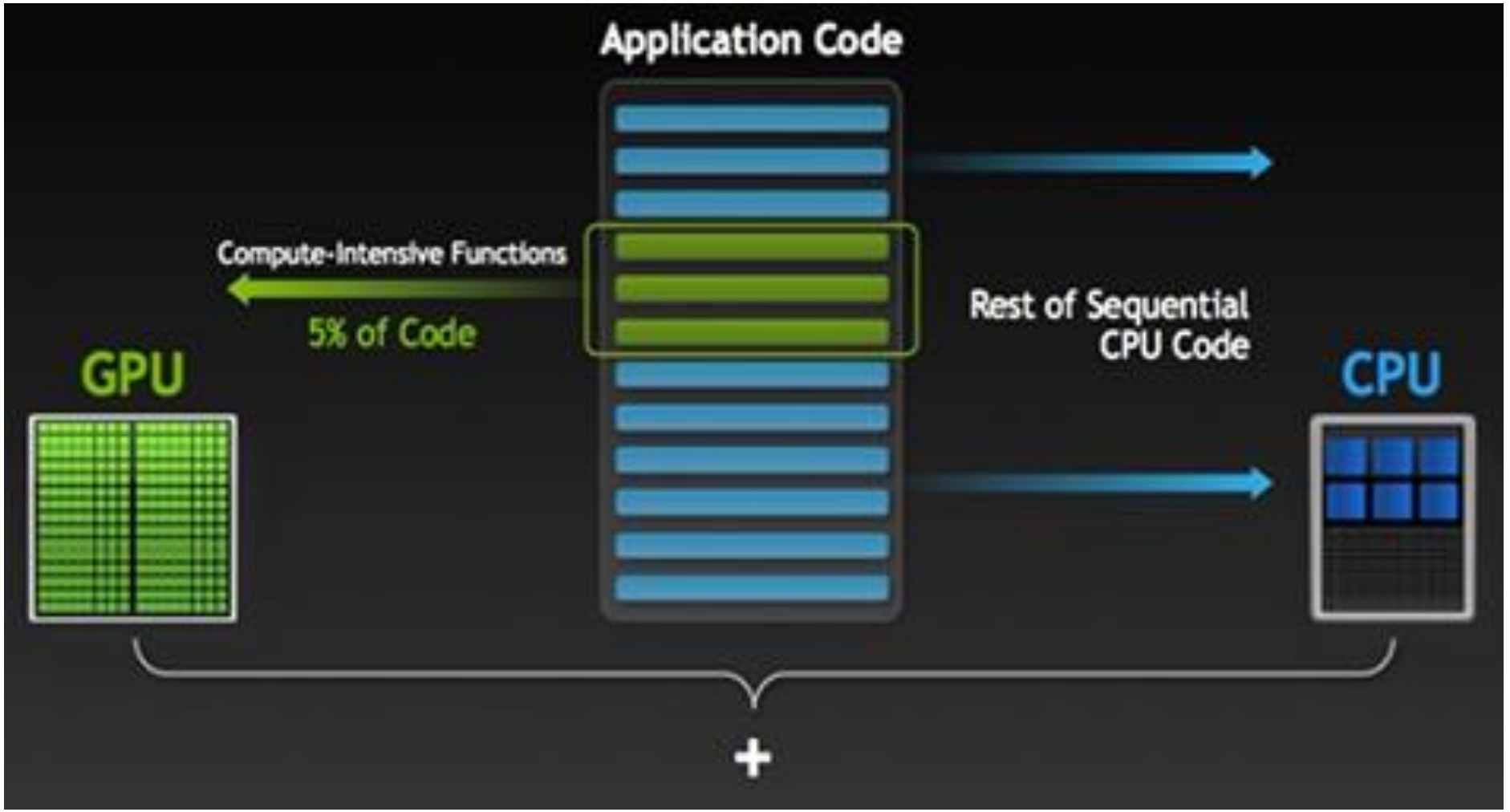
GPU and CPU Communications

- There are 2 problems in this this GPU and CPU communication
  - If this command buffer is empty, then we have limited CPU and the GPU will spin around waiting for new input.
    - This means that the GPU will not make any difference and the processing of the application will not be faster.
  - On the other hand if the command buffer is full, the CPU will spin around waiting for the GPU to consume it and we have limited GPU capability

GPU Computing

- GPU computing or GPU-accelerated computing is the use of graphics processing units together with central processing units to accelerate the scientific and enterprise application processing.
- The computationally and data intensive parts that can be broken down into smallr parts are parallelised and processed on the GPUs while the serial parts of the computations are done on the CPUs

How GPU Accelerates Computation



Source: NVIDIA

Why Are GPUs More Efficient Than CPUs?

- GPUs have more arithmetic logic units (ALUs) than CPUs
- GPUs allocates more ALUs and transistors to data processing than memory management

GPU Programming

- GPU Programming is the at of writing parallel programs that run on GPUs using a GPU-enabled and compliant platform and language such as NVIDIA CUDA C/Fortan or OpenACC
- There are 3 ways to programme GPUs
  - Use GPU-Accelerated Libraries
    - There are already written and tested libraries that programmers can just pick and use in their programmes
  - Use GPU Directives
    - You can use automatic parallel loops in your C or Fortran programme using GPU directives such as OpenACC directives
  - Develop your own programme
    - You can programme GPUs by writing your own GPU-enabled codes using one of your preferred high level programming languages such as C or C++
      - Use Compute Unified Device Architecture (CUDA) in conjunction with these languages such as CUDA C/C++

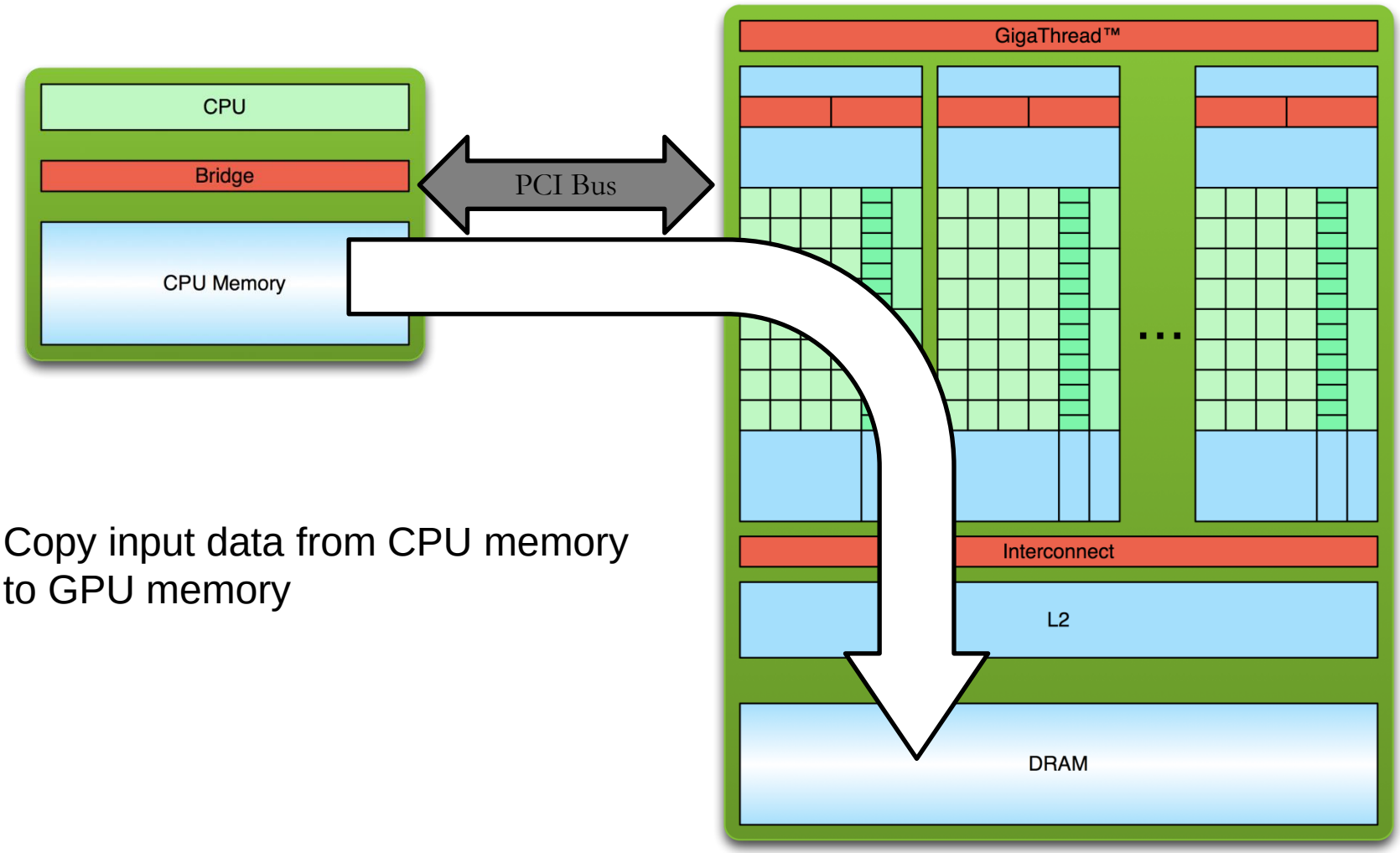
Memory in GPU

- Device Memory
  - This resides in the CUDA address space
  - May be accessed by CUDA kernels through normal C/C++ pointer and array dereferencing operations
  - Most GPUs have dedicated pool of device memory is directly attached to the GPU and accessed by a memory controller
  - All memory allocations are backed by actual physical memory and no virtual memory allocation as in CPU
    - This means that when the physical memory of GPU is exhausted, memory allocations will fail.

Memory in GPU

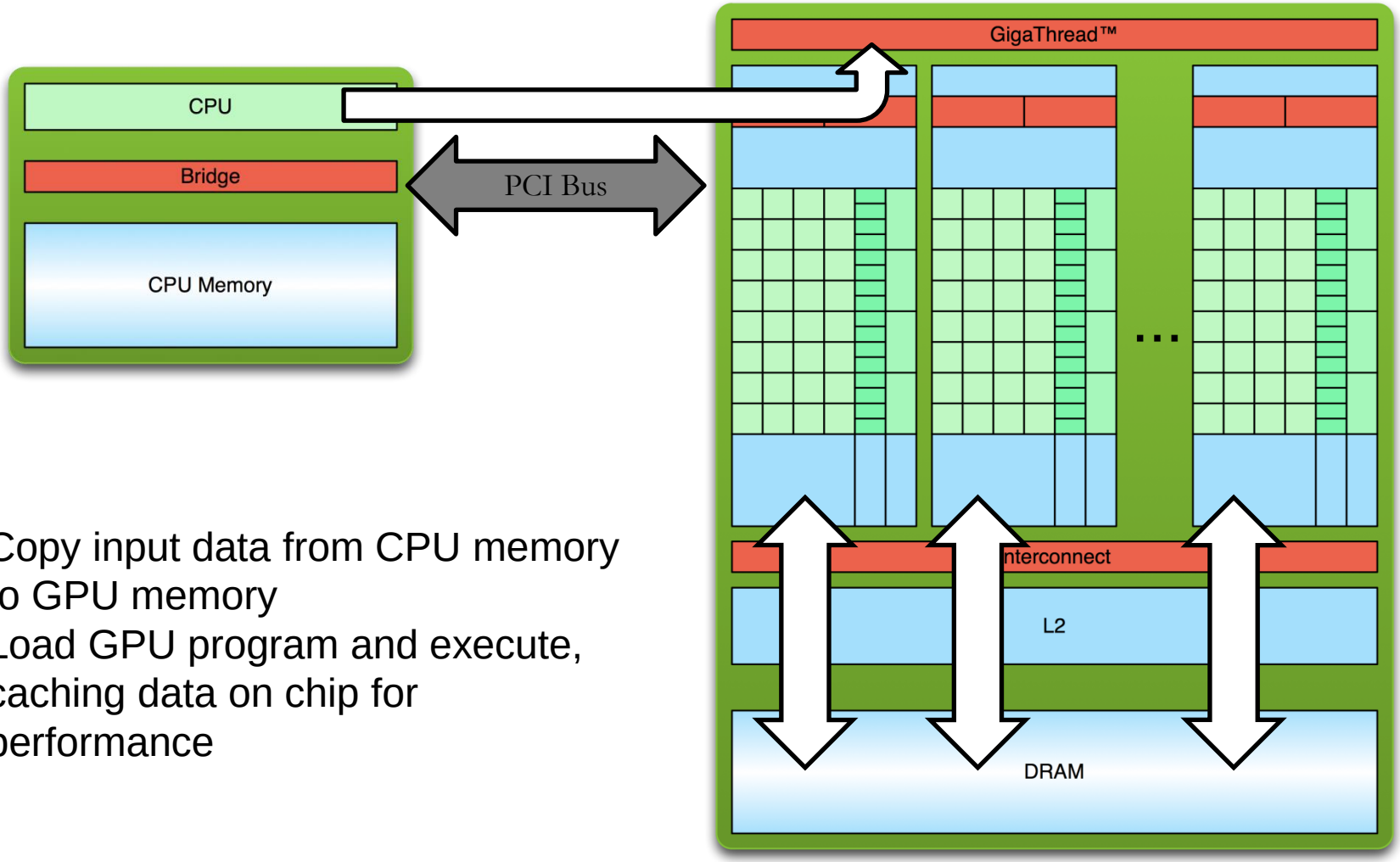
- Host Memory
  - This is CPU memory
  - It is managed by malloc(), free(), new[], delete[] built in functions
  - In CUDA, host memory is virtualised
    - This means that when the physical memory is exhausted, pages of virtual maemory can be reallocated without changing their virtual addresses
  - The operating system that manages virtual memory is called “Virtual Memory Manager” or VMM.
  - GPUs can read or write host memory using the facility called “Direct Memory Access” or DMA
  - DMA enables GPU hardware to work concurrently with CPU

Simple Processing Flow



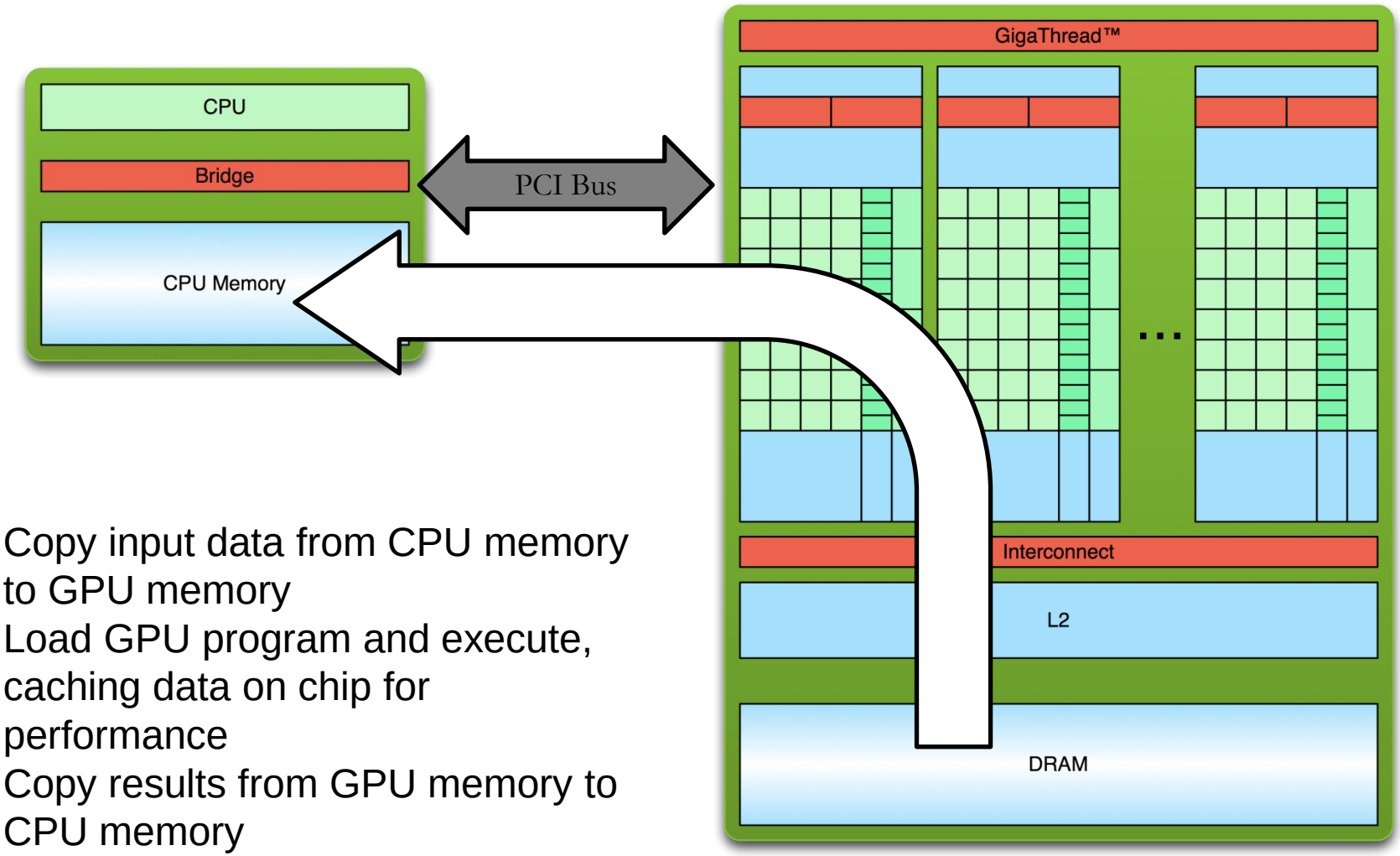
1. Copy input data from CPU memory to GPU memory

Simple Processing Flow



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

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

Advantages of using GPUs

- Faster processing speed
- Energy efficiency
- Simplified design
- Less communication cycles

Class Task

- Name some companies and institutions that are likely to use GPUs
- What do you think are the limitations and challenges of implementing GPUs