Plan for Integrating TOUCH Modules in CS516 Parallelization of Programs

Vishwesh Jatala
Assistant Professor
Department of EECS
Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

Course Overview

- Parallelization of Programs
- Senior UG/PG Level
- 6 credit course ~ 40 hours class
- August - November end

Course Overview: Original

- Introduction to parallelization;
- Performance; Amdahl's law;
- Techniques for extracting parallelism from sequential programs
- Compile-time parallelization
- Runtime parallelization
- Synchronization
  - Scheduling techniques;
  - Parallelization for cache performance;

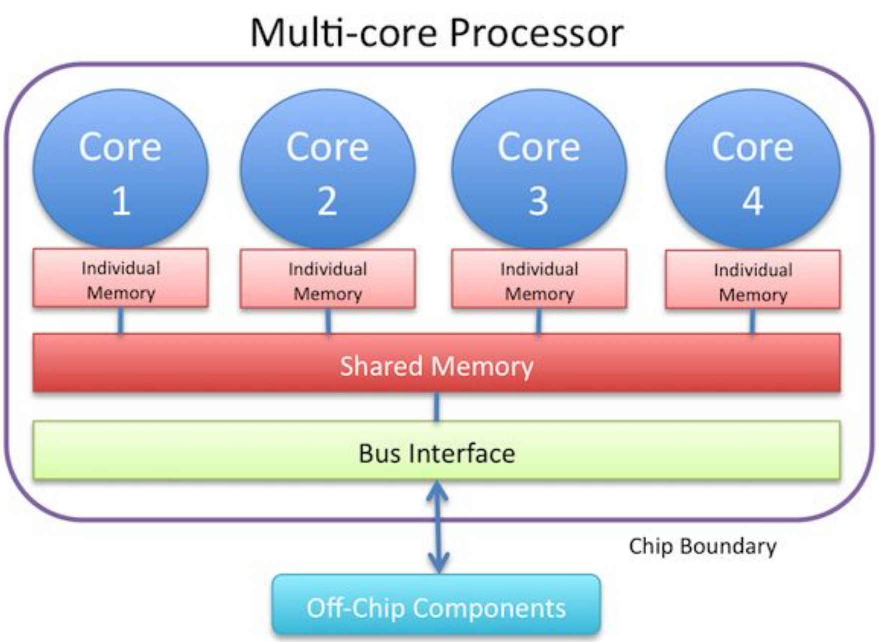
Course Overview: Integration

- Introduction to parallelization;
- Performance; Amdahl's law;
- Touch Modules D1, C2, and B2
  - Parallel Hardwares -- GPUs.
  - Introduction to CUDA programming
  - Google Colab
  - Instruction Execution in GPUs
- Techniques for extracting parallelism from sequential programs
- Compile-time parallelization
- Runtime parallelization
- Synchronization
  - Scheduling techniques;
  - Parallelization for cache performance;

Motivation

- For many decades, the single core processors were popular
  - Instruction-level parallelism
  - Core clock frequency
  - Moore's law
- Mid-to late-1990s - power wall
  - Power constraints
  - Heat dissipation
- Multicore processors, accelerators, such as GPUs.

Why GPUs?

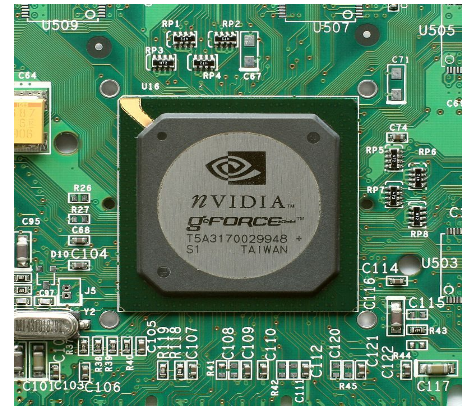


- Multicore processors
  - Task level parallelism
  - Graphics rendering is computationally expensive
  - Not efficient for graphics applications

Images Source: Internet

Graphics Processing Units

- The early GPU designs
  - Specialized for graphics processing only
  - Exhibit SIMD execution
  - Less programmable

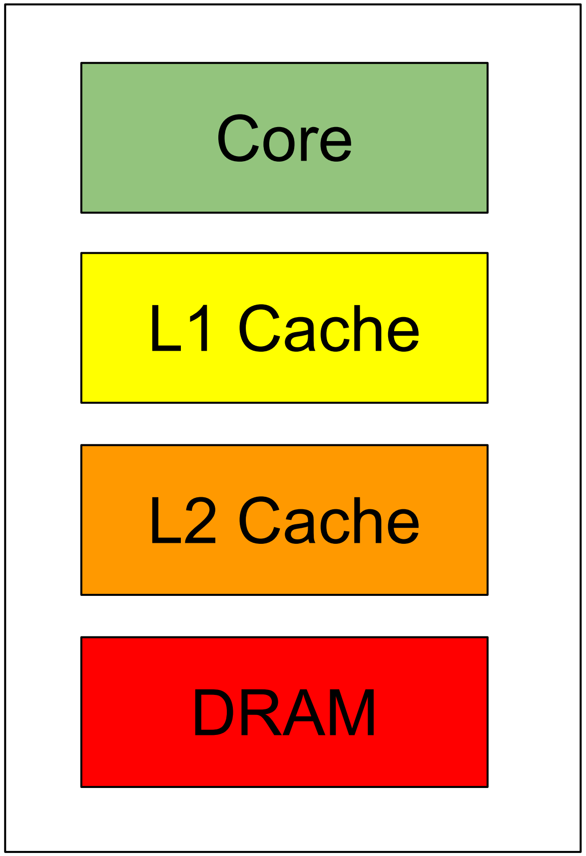


NVIDIA GeForce 256

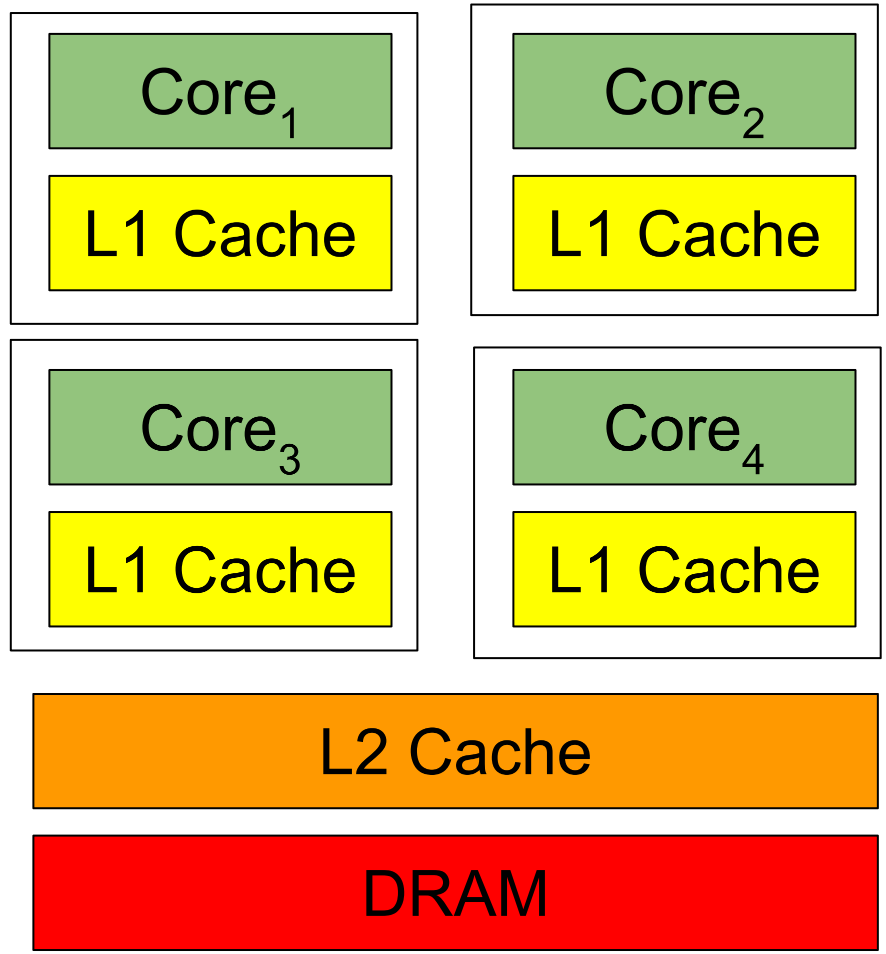
- In 2007, fully programmable GPUs
  - CUDA released

Images Source: Internet

Single-core CPU vs Multi-core vs GPU

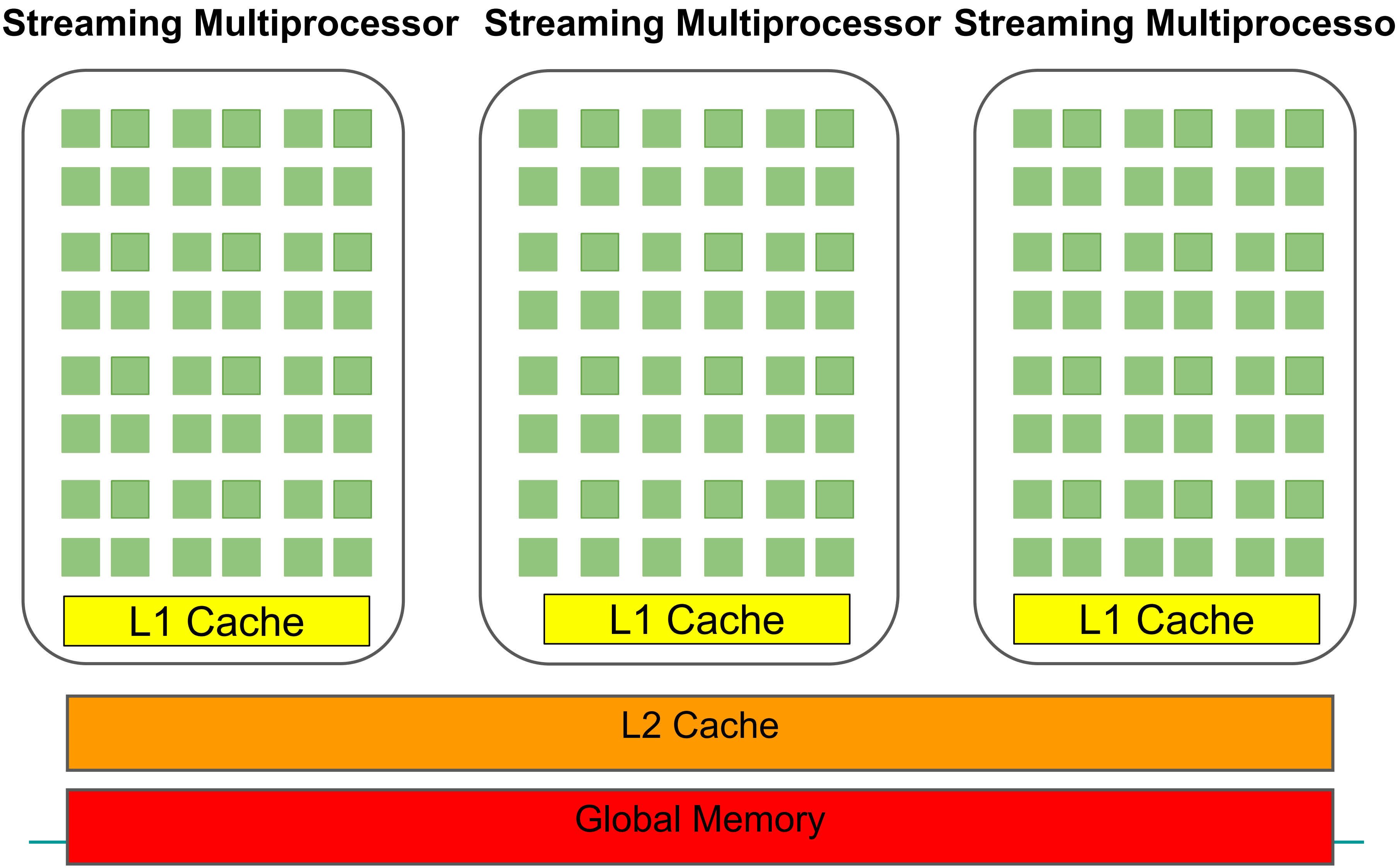


Single-core CPU

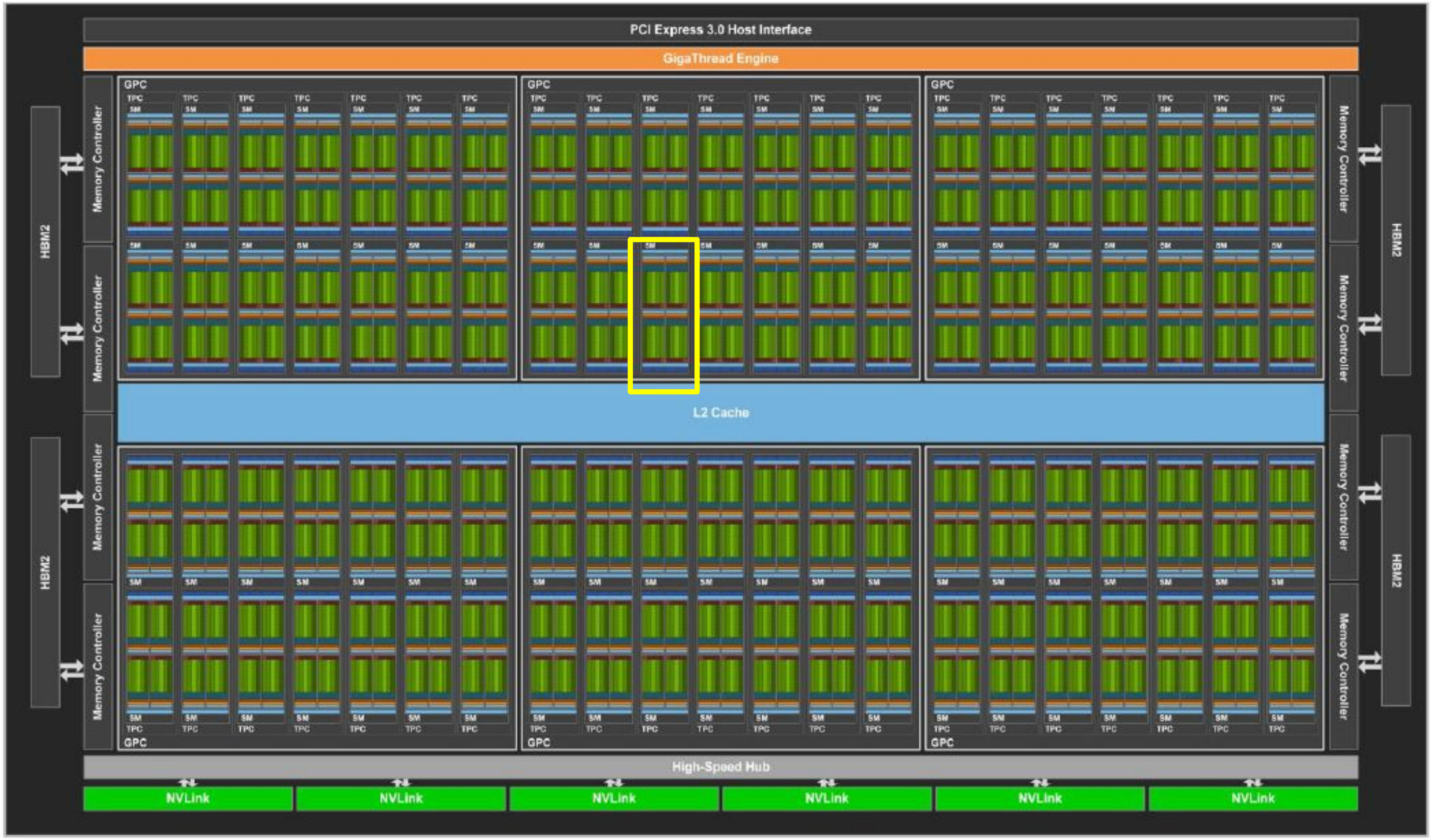


Multi-core CPU

Single-core CPU vs Multi-core vs GPU

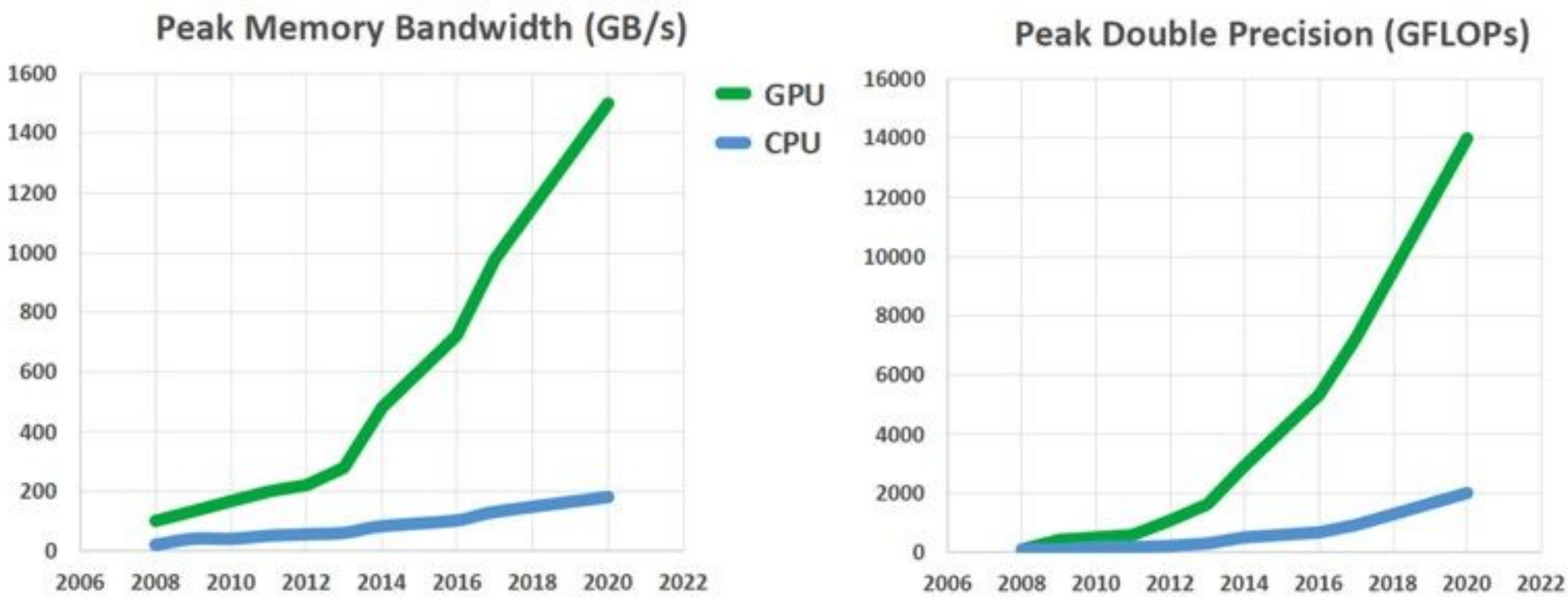


NVIDIA Volta GV100



https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

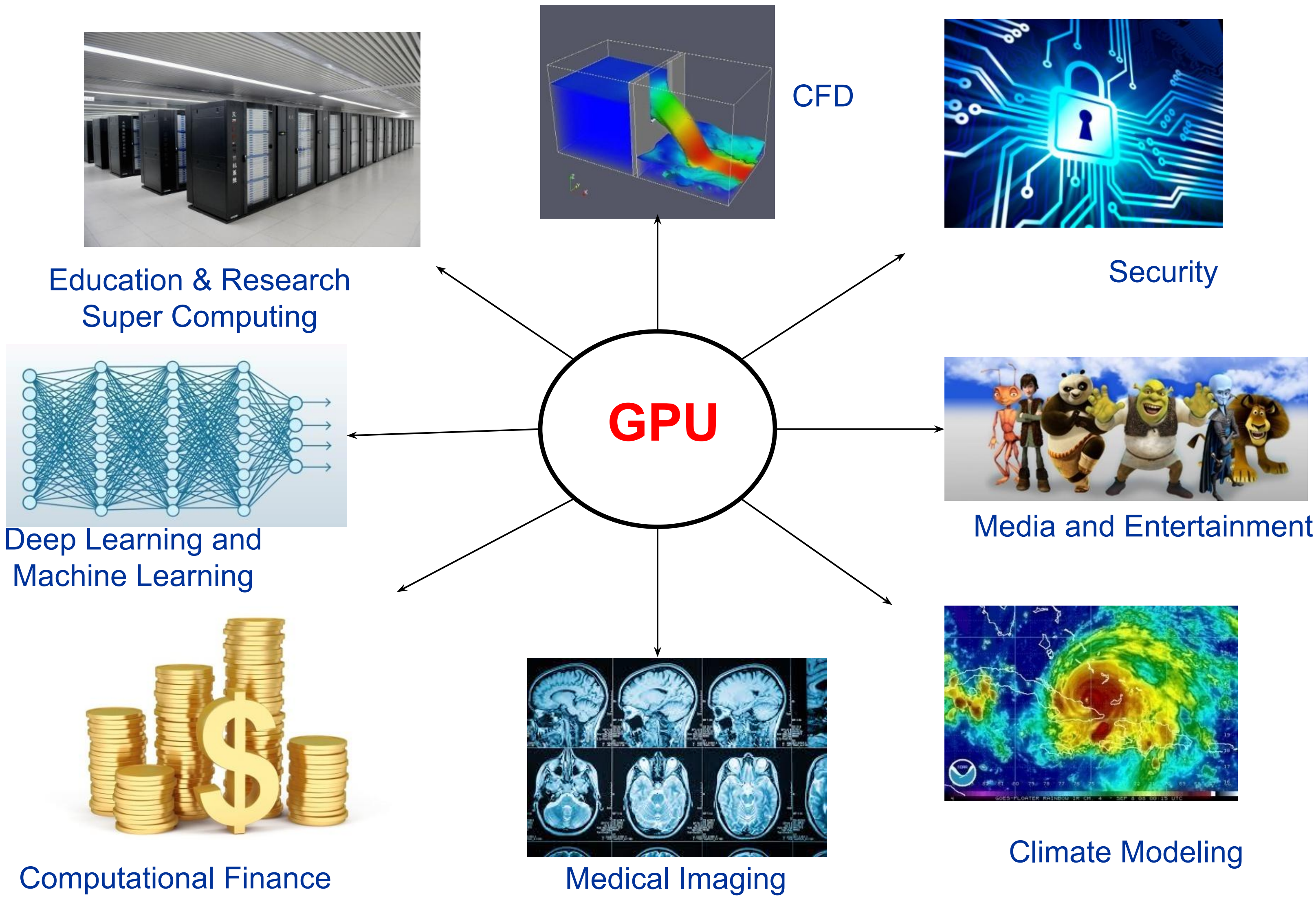
CPU vs GPU



Chip to chip comparison of peak memory bandwidth in GB/s and peak double precision gigaflops for GPUs and CPUs since 2008.

https://www.nextplatform.com/2019/07/10/a-decade-of-accelerated-computing-augurs-well-for-gpus

GPU Applications



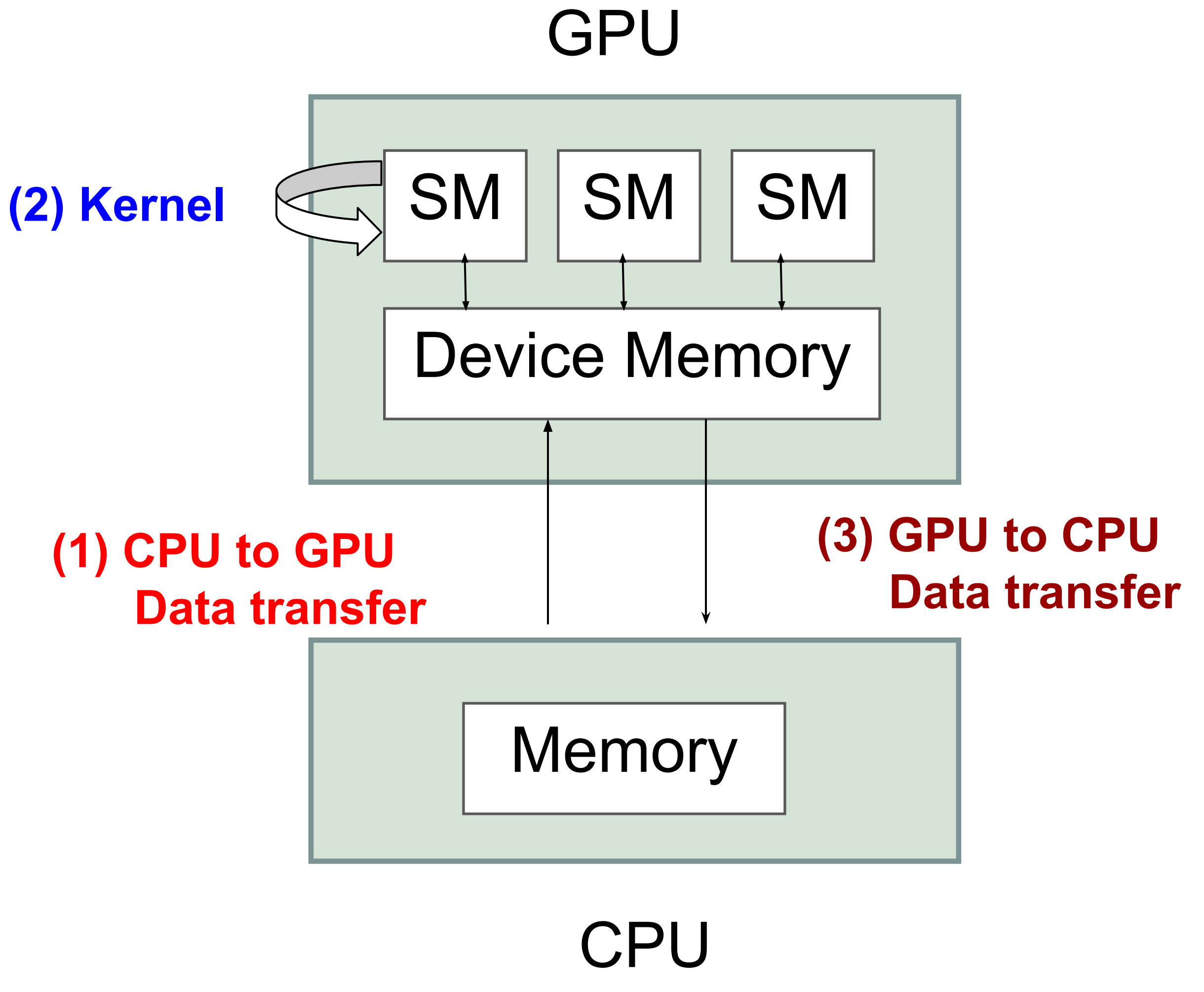
Reference: https://www.nvidia.com/
Images Source: Internet

Programming for GPUs

Programming Models

- CUDA (Compute Unified Device Architecture)
  - Supports NVIDIA GPUs
  - Extension of C programming language
  - Popular in academia

Introduction to CUDA Programming



Simple Example: Touch Module Example

```
__global__ void Vector_Add(float a, float b, float c){
      /* Compute index i based on thread id */
      c[i] = a[i] + b[i];
}
int main() {
        h_a = malloc(..)//host array
        cudaMalloc(d_a,….) //device

         /* Initialize h_a, h_b, h_c */

        cudaMemcpy(d_a, h_a, cudaMemcpyHostToDevice)
       //Similarly do for d_b, and d_c
        Vector_Add<<<ThreadConfig>>> (d_a, d_b, d_c);

        cudaMemcpy(h_c, d_c, cudaMemcpyDeviceToHost)
        process(h_c);
        cudaFree(d_a);cudaFree(d_b);
       cudaFree(d_c);
        free(h_a);
}
```

Compute Kernel (points to the `__global__ void Vector_Add(...)` line)

CPU to GPU Data transfer (points to `cudaMemcpy(d_a, h_a, cudaMemcpyHostToDevice)`)

Invoke Kernel (points to `Vector_Add<<<ThreadConfig>>> (d_a, d_b, d_c);`)

GPU to CPU Data transfer (points to `cudaMemcpy(h_c, d_c, cudaMemcpyDeviceToHost)`)

Threadblock configuration: Touch Module Example

Vector_Add<<<ThreadBlocks, Threads> (d_a);

- Thread block configuration
  - User choice
  - Depends on problem size
- Problem size = 32768 (1024 * 32)
  - Threadblocks = 32, No of threads/thread block = 1024
  - Threadblocks = 128, No of threads/thread block = 256

CUDA thread block occupancy:
https://docs.nvidia.com/cuda/cuda-occupancy-calculator/index.html

Demo using Colab on Touch based Example

Evaluation

- Evaluation scheme:
  - Tierce exam-1: ~12.5%
  - Tierce exam-2: ~12.5%
  - Programming assignments (4-5): ~20% (D1 and C2 Modules)
    - CUDA Programming assignments: Similar to Matrix multiplication in Google Colab
  - Projects: ~40% (B2 Module)
    - Ideas: Parallelizing several algorithms for GPUs using CUDA.
      - Community detection
      - Graph mining applications
      - Deep learning
      - SpMV and SPMM.
  - Quiz: ~5%

Thank You!