



CE/CZ 3001: Advanced Computer Architecture

Module 6: GPU Architecture and CUDA Programming

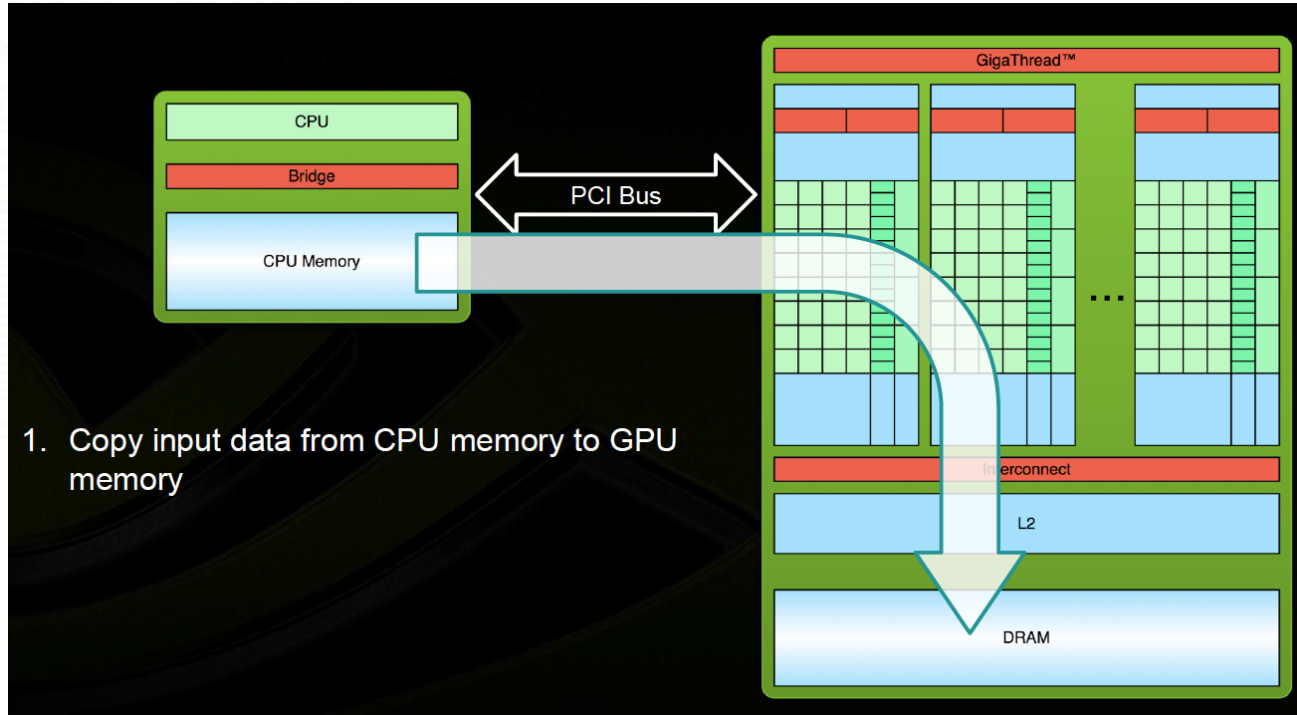
- GPU architecture and CUDA programming

Asst Prof Liu Weichen
School of Computer Science and Engineering
Nanyang Technological University, Singapore

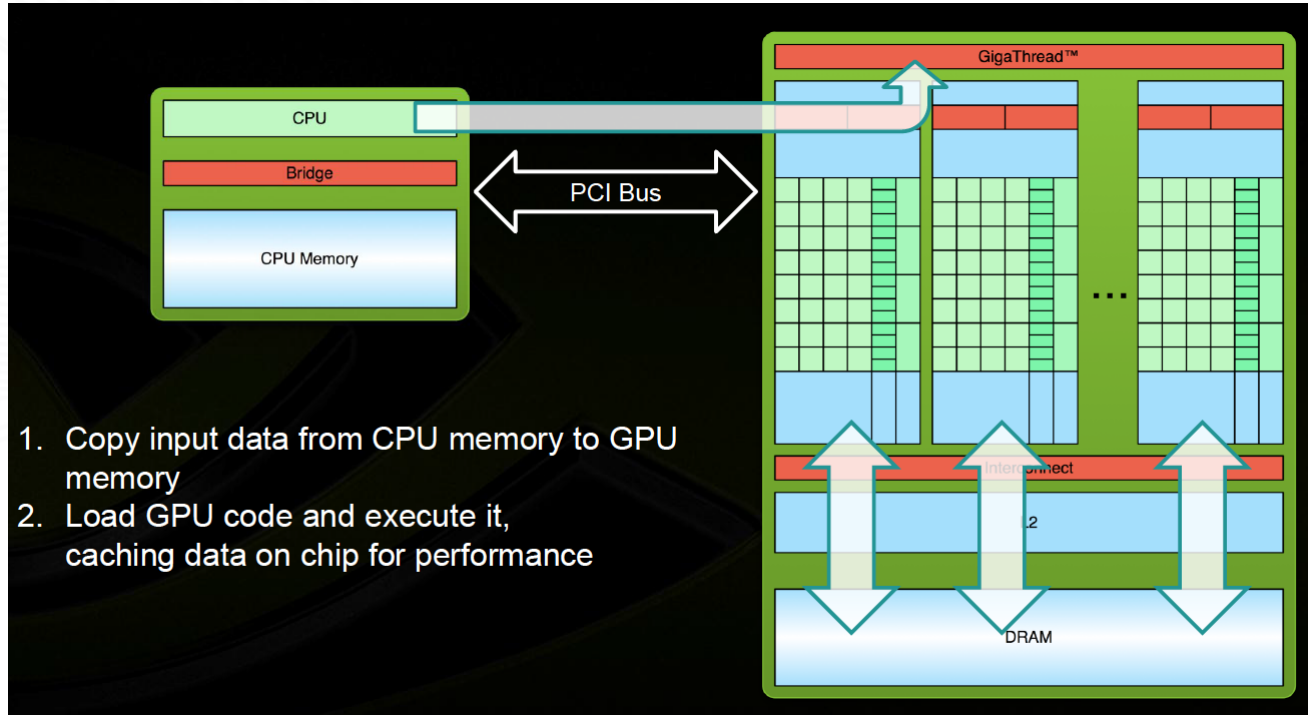
Summary

- Basic GPU architectures
- Kernels, threads and thread blocks
- Sync and memory management between host and device
- Data sharing and sync between threads

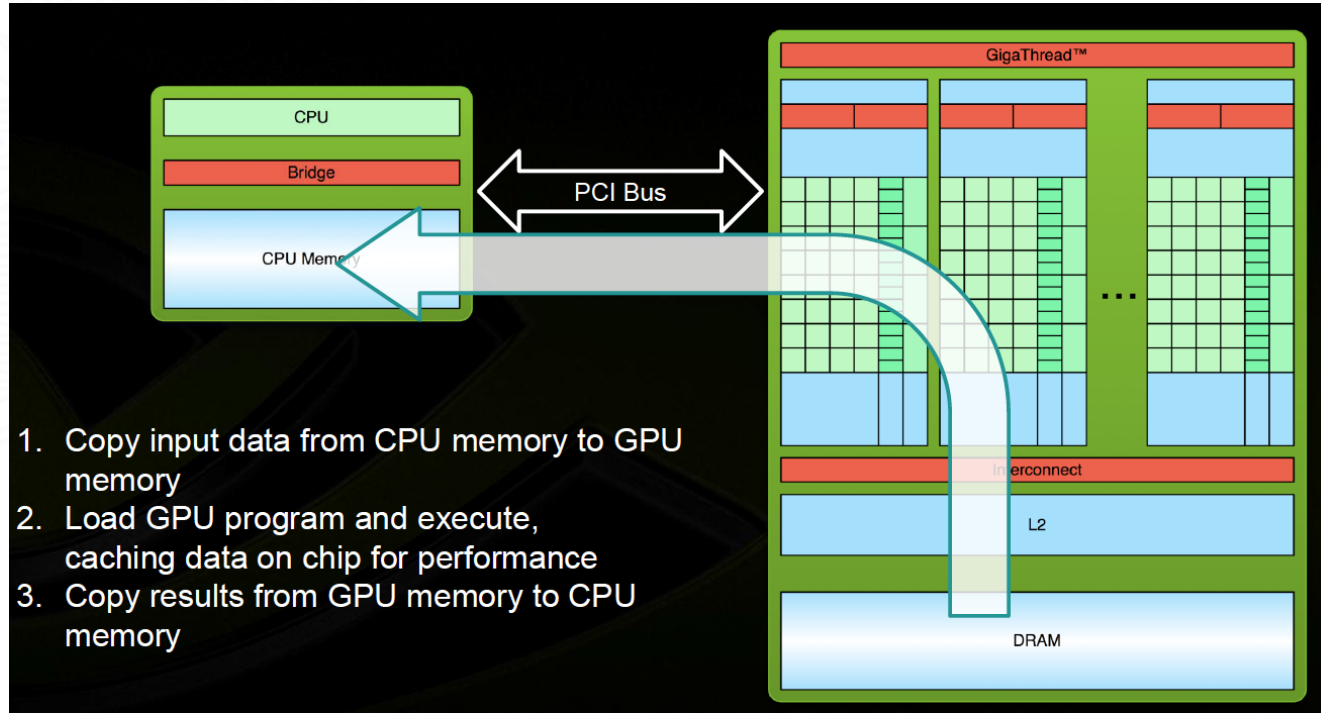
Simple Processing Flow of Heterogeneous Computing (1)



Simple Processing Flow of Heterogeneous Computing (2)



Simple Processing Flow of Heterogeneous Computing (3)



CUDA programming basics (1)

- Difference between host and device
 - Host CPU
 - Device GPU
- Using `__global__` to declare a function as device code
 - Executes on the device
 - Called from the host
- Passing parameters from host code to a device function
- Basic device memory management
 - `cudaMalloc()`
 - `cudaMemcpy()`
 - `cudaFree()`

CUDA programming basics (2)

- Launching parallel threads
 - Launch N blocks with M threads per block with `kernel<<<N,M>>>(...);`
 - Use `blockIdx.x` to access block index within grid
 - Use `threadIdx.x` to access thread index within block
- Allocate elements to threads:
 - `int index = threadIdx.x + blockIdx.x * blockDim.x;`
- Use `__shared__` to declare a variable/array in shared memory
 - Data is shared between threads in a block
 - Not visible to threads in other blocks
- Use `__syncthreads()` as a barrier
 - Use to prevent data hazards

Excise 1

Which of the following are design philosophies in GPU architecture?

- (1) GPUs use massive parallelism to hide stalls
- (2) GPUs have many low-latency execution units
- (3) GPUs spread the cost of managing an instruction stream across many ALUs

- (A) 1 and 2
- (B) 1 and 3
- (C) 2 and 3
- (D) all of the above

Excise 2

If we want to allocate an **array of float** in the GPU global memory and have the pointer variable **device_array** to point to the array, what is the correct call to **cudaMalloc()** on the host side?

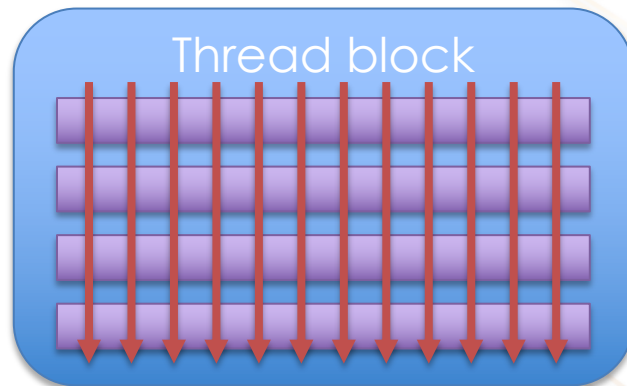
```
// pointers to host & device arrays  
float *device_array;  
int size = num_bytes * sizeof(float);
```

- (A) `cudaMalloc(device_array, size);`
- (B) `cudaMalloc((void *)device_array, size);`
- (C) `cudaMalloc((void *)&device_array, size);`
- (D) `cudaMalloc((void **)&device_array, size);`

Excise 3

We want to use each **thread** to calculate **four output elements of a vector addition**. Each thread block processes four sections. All threads in each block will first process a section first, each processing one element. They will then all move to the next section, with each thread processing one more element. This repeats until all four sections are processed. Assume that variable i should be the index for the first element to be processed by a thread. What would be the expression for mapping the thread/block indices to data index of the first element?

- (A) $i = \text{blockIdx.x} * \text{blockDim.x} * 4 + \text{threadIdx.x};$
- (B) $i = \text{blockIdx.x} * \text{threadIdx.x} * 2;$
- (C) $i = (\text{blockIdx.x} * \text{blockDim.x} + \text{threadIdx.x}) * 4;$
- (D) $i = \text{blockIdx.x} * \text{blockDim.x} * 2 + \text{threadIdx.x};$



Excise 4

Can we put `__syncthreads()` in the if-statement as follows?

```
if (t < BLOCK_SIZE / stride) {  
    __syncthreads();  
    partialSum[t*stride*2] += partialSum[t*stride*2+stride];  
}
```

No. Because if there exists control divergence in a warp, the threads which satisfy the if-statement will be in active and wait for the threads which do not satisfy the if-statement forever because these threads are deactivated.

Excise 5

Write some CUDA kernel code in which the threads in the same block transpose the elements of a matrix `mat` in shared memory in-place. That is, all threads should see that `mat[i][j]` is swapped with element `mat[j][i]` after the transposition. You can assume that each block has dimensions `dim3(16,16,1)`. Make your code as efficient in time and space as possible.

```
__shared__ float mat[16][16];
```

```
float temp = mat[threadIdx.y][threadIdx.x];  
__syncthreads();  
mat[threadIdx.x][threadIdx.y] = temp;  
__syncthreads();
```