

Lecture 12-1

Multi-GPUs

Getting more power!

* Precision Issue

⇒ Solution

① error bounds (10e-5)

② rounding method

정확도 문제
 (e.g.) -- fadd -- [rn, rn, rn, rn] (x, y)
 sub
 mul
 ...

Hw로
 문제 발생!

Fused Multiply-Add (FMA)

⇒ without FMA
 → rn(rn(x*y)+z)

⇒ Compute (x*y+z)

→ rn(x*y+z)

precision issue

optim (Hw로 round error)

(compiler error)

* release mode

compiled error

⇒ FMA가 필요!

왜 400 이상

⇒ CPU와 GPU의 연산 방식 차이

NVCC option

⇒ -sm=sm_52

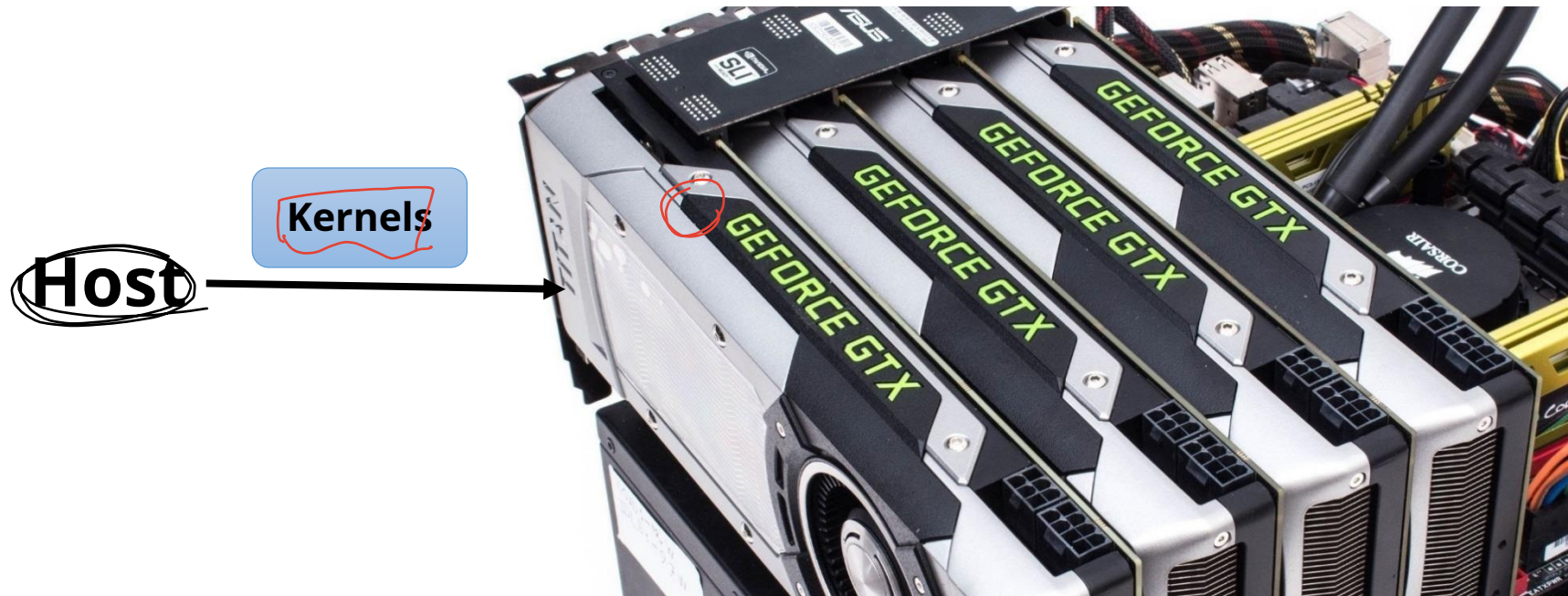
⇒ FMA 사용 X

⇒ CPU와 GPU의 연산 방식 차이

⇒ CPU 연산이 더 정확함
 연산 후 "rn"으로 round 하기 (반올림)

How Many GPUs Do You Have?

How Many GPUs Can You Use?



Getting Information of Your System

- The number of CUDA-enabled devices

- cudaError_t cudaGetDeviceCount (int *count);

GPU count return.

- Querying properties of each device

- cudaError_t cudaGetDeviceProperties
(cudaDeviceProp* prop, int deviceID)

- cudaDeviceProp structure

- See the programming guide [\[link\]](#)

0, 1, ...

return

```
struct cudaDeviceProp {  
    char name[256];  
    cudaUUID_t uuid;  
    size_t totalGlobalMem;  
    size_t sharedMemPerBlock;  
    int regsPerBlock;  
    int warpSize;  
    size_t memPitch;  
    int maxThreadsPerBlock;  
    int maxThreadsDim[3];  
    int maxGridSize[3];  
    int clockRate;  
    size_t totalConstMem;  
    int major;  
    int minor;  
    size_t textureAlignment;  
    size_t texturePitchAlignment;  
    int deviceOverlap;  
    int multiProcessorCount;  
    ...  
};
```

Query GPU Information

f typing @

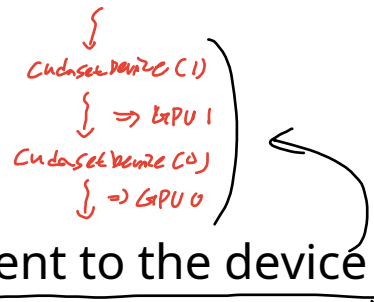
```
void main(void) {  
    int ngpus;  
    cudaGetDeviceCount(&ngpus);  
  
    for (int i = 0; i < ngpus; i++) {  
        cudaDeviceProp devProp;  
  
        cudaGetDeviceProperties(&devProp, i);  
        printf("Device[%d](%s) compute capability : %d.%d.\n"  
              , i, devProp.name, devProp.major, devProp.minor);  
    }  
}
```

Device[0](GeForce RTX 2080 Ti) compute capability : 7.5.
Device[1](GeForce GTX 1080) compute capability : 6.1.

Selecting the target GPU

- Set the target device

- cudaError_t **cudaSetDevice** (int deviceID)
- After calling this function, all operations are sent to the device



- Get the current device ID

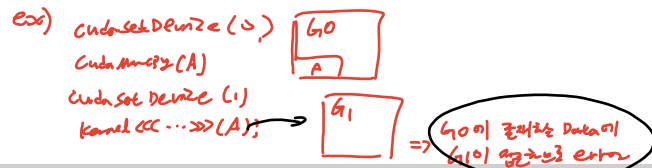
- cudaError_t **cudaGetDevice** (int *deviceID)

→ return.

- We can change the target device any time

- However, we should carefully changing the target device

- Which device having the target data?



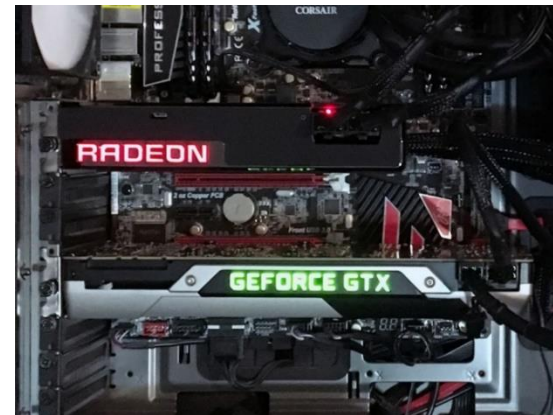
Lecture 12-2

Heterogeneous Parallel Computing

Fully utilize your computing resources

Heterogeneous Architecture

- A heterogeneous architecture consisting of more than one type of computing resources
- Examples
 - A desktop PC having both multi-core CPUs and GPUs
 - A multi-GPU system consisting of different types of GPUs



Heterogeneous Computing

- Use multiple heterogeneous computing resources at once for solving a problem

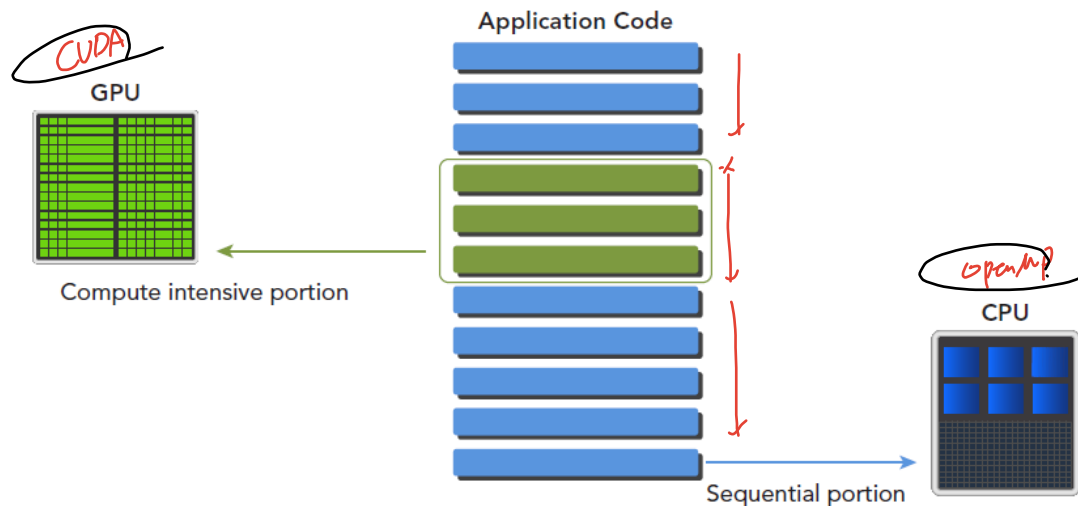
- \leftrightarrow Homogeneous computing \rightarrow $\frac{1}{n}$ times

- Advantage

- Fully utilize all available computing resources
 - Achieve high performance



Heterogeneous Computing



[Figures from Professional CUDA C Programming, Wrox]

Host-Device Concurrent Execution



[Image from PublicDomainPictures, Pixabay]

Example – OpenMP + CUDA

```
#pragma omp parallel for CPU Thread
LOOP_I(NUM_STREAMS)
{
    // Do work of device
    GPU { cudaSetDevice(device[i]);
        myKernel <<< NUM_BLOCK / NUM_STREAMS, NUM_T_IN_B, 0, stream[i]>>> (...);
        GPU work
    }

    // Do work of host
    CPU { for(...) {
        }
        CPU work

        // Synchronization
        cudaStreamSynchronize(stream[i]);
    }
}
```

Diagram illustrating thread execution and synchronization:

T_1 and T_2 threads are shown. T_1 executes GPU work (G_0) and then CPU work (S_0). T_2 executes GPU work (G_1) and then CPU work (S_1). Arrows indicate the flow of execution and synchronization points.

Handwritten notes in the code:

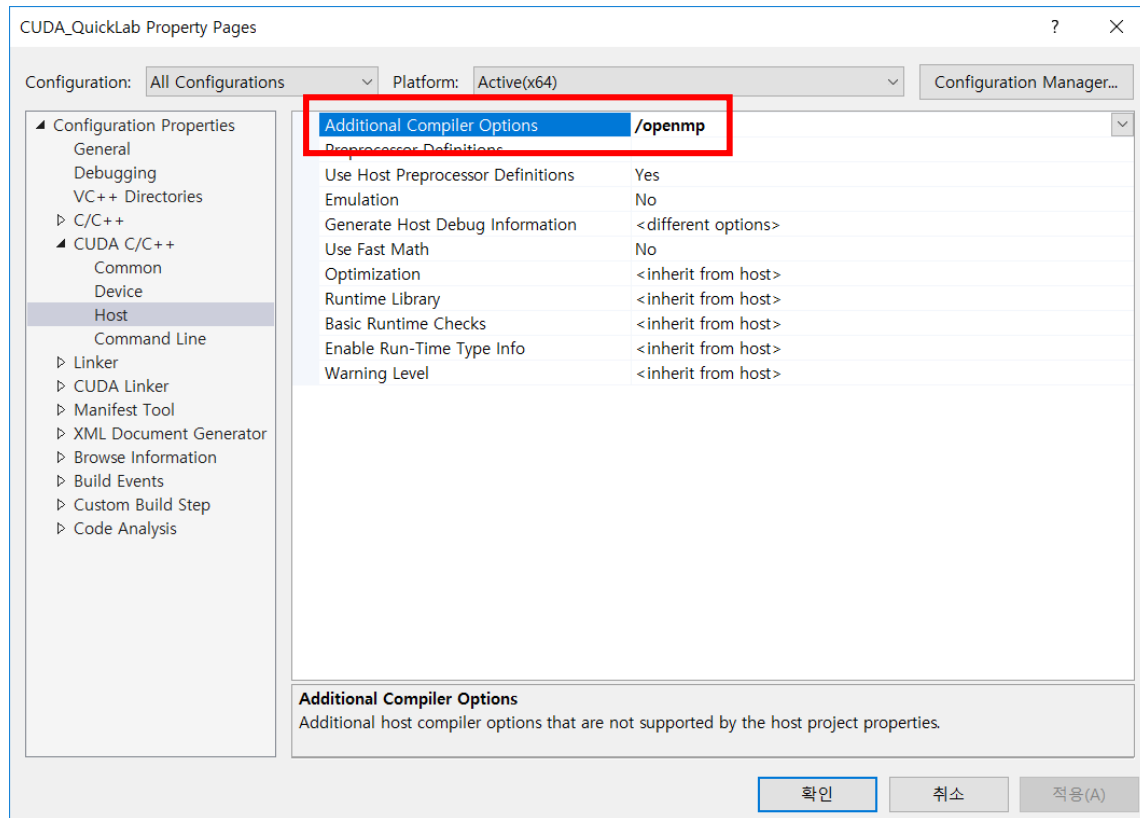
- \Rightarrow CPU work GPU work
- \Rightarrow CPU work GPU work

Using Both OpenMP and CUDA

• NVCC compiler option

- Using OpenMP on CUDA code (.cu)

→ compile options Add



Using Both OpenMP and CUDA

- Decouple source file for host code and device code
 - See the CUDA MatrixAdd Project on the course git repository [[link](#)]

