CUDA 프로그래밍

https://youtu.be/PE2p10N11gg

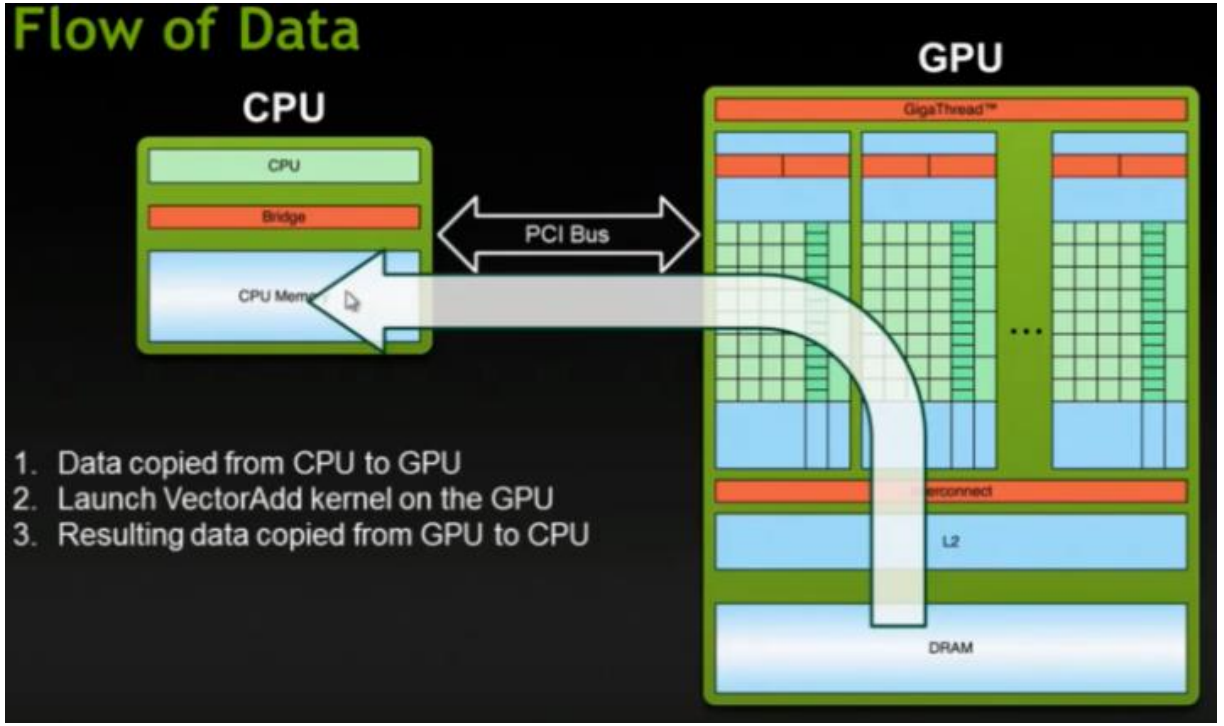
CUDA

- CUDA는 Nvidia가 만든 병렬 컴퓨팅 플랫폼 및 API 모델이다.

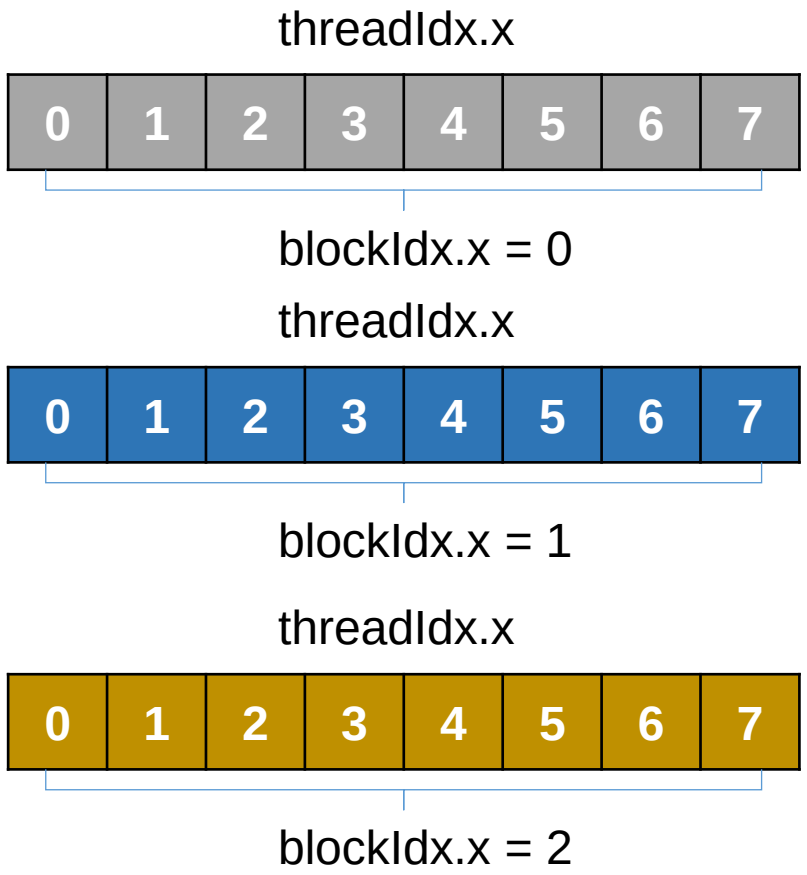
- CUDA 플랫폼은 GPU 가상 명령어셋과 병렬 처리 요소들을 사용할 수 있도록 만들어주는 소프트웨어 레이어이며, Nvidia가 만든 CUDA 코어가 장착된 GPU에서 작동한다.

CUDA 데이터 흐름

- CUDA를 이용한 GPU 프로그래밍 처리 과정
  1. Data가 CPU에서 GPU로 복사
  2. GPU에서 커널 함수를 실행하여 처리
  3. 결과를 GPU에서 CPU로 복사



CUDA 구조

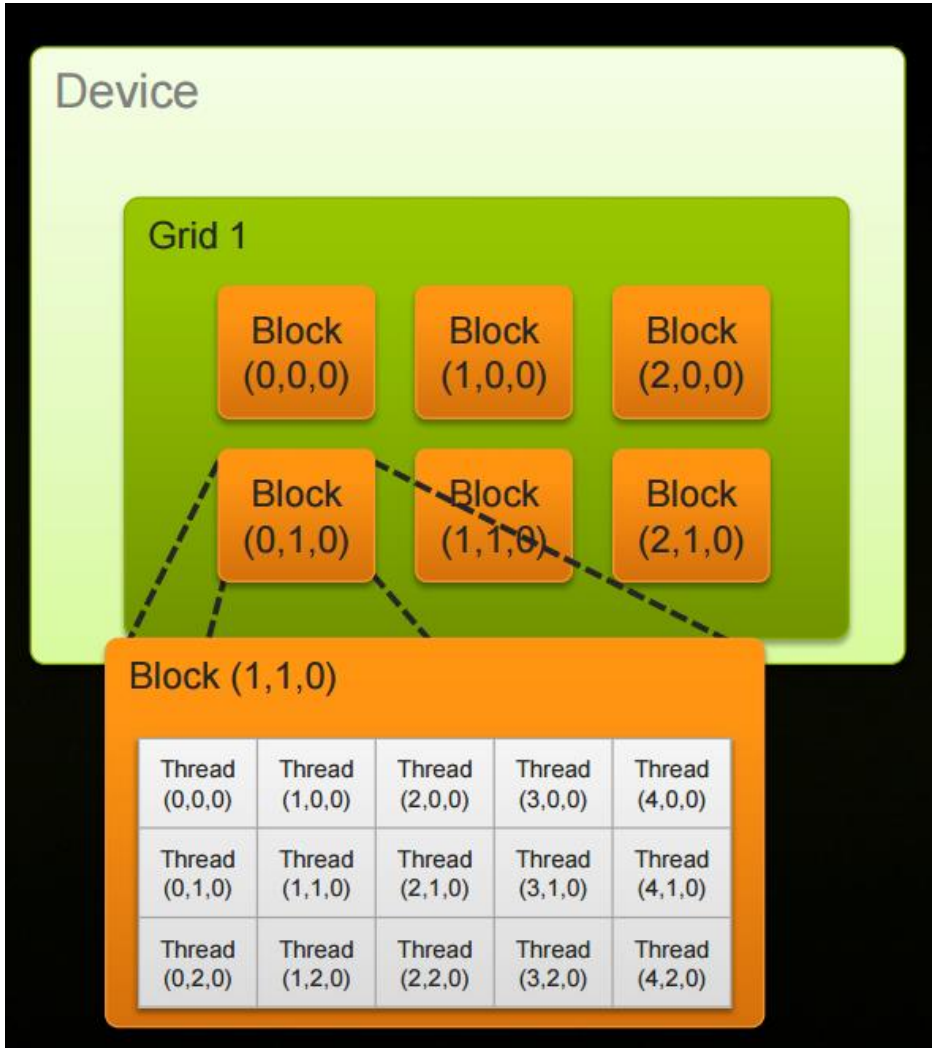


Grid

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

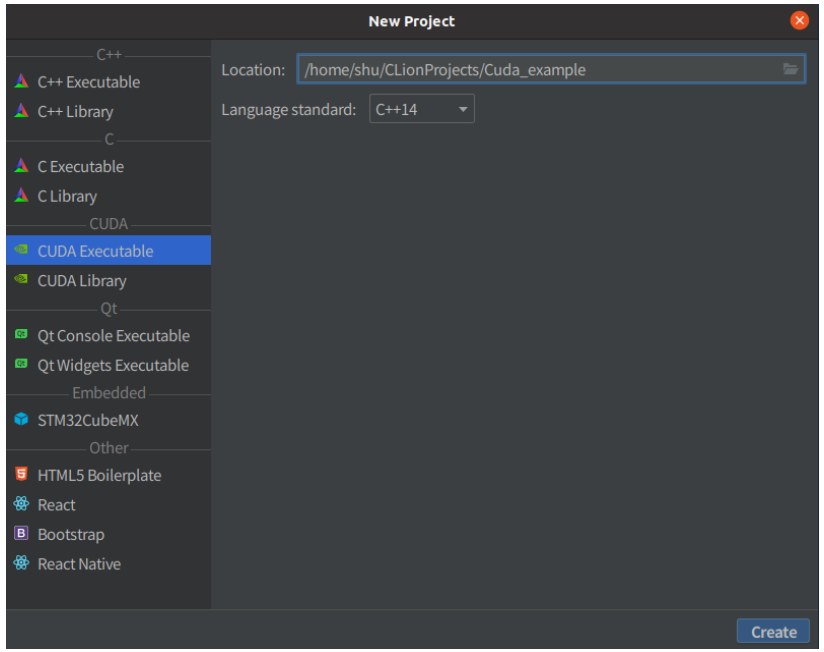
CUDA 구조

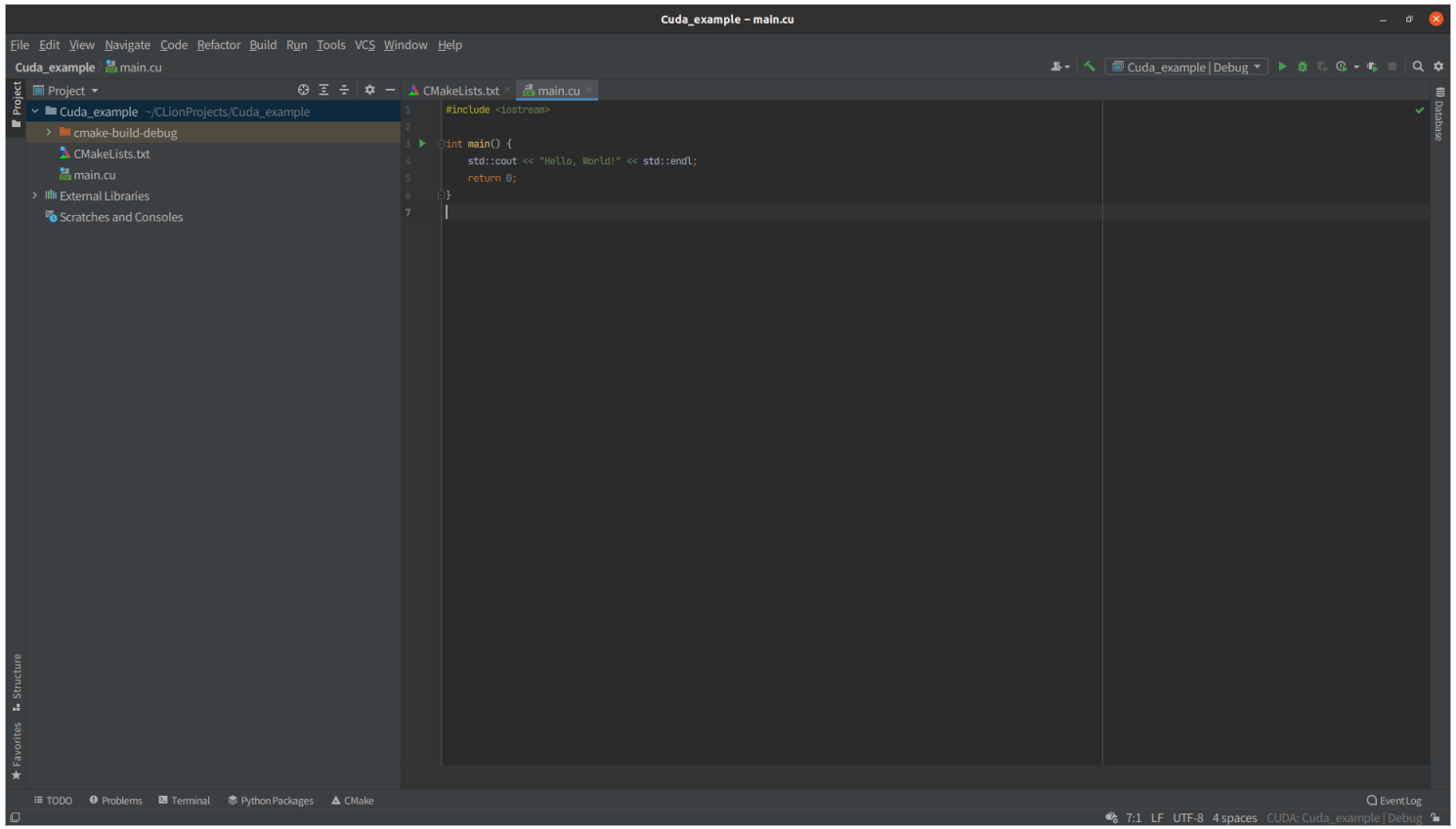
- Variables
  - threadIdx : block 안에서의 thread 번호
  - blockIdx : block 번호
  - blockDim : block당 thread 개수
  - gridDim : block의 개수



CUDA 간단한 예제

실습 환경 : 우분투 20.04 / Clion IDE





CUDA 간단한 예제

```c
#include <cstdio>

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

```c
#include <cstdio>

__global__ void kernel(void){

}

int main(){
    kernel<<< gridDim: 1, blockDim: 1>>>();
    printf( format: "Hello, World\n");
    return 0;
}
```

- Host : CPU and Host memory
- Device : GPU and Device memory

- 디바이스 코드는 nvcc(Nvidia Compiler)가 컴파일 하는데, 키워드가 필요함
  - Nvcc : source file을 host components와 device components로 나눈다.
  - __global__ : Host를 통해 호출되어 Device에서 작동되는 함수
  - Kernel() -> device component
  - Main() -> host component

CUDA 키워드

```
__device__ int deviceAdd(){

}

__host__ int hostAdd(){

}

__device__ __host__ int devhostAdd(){

}

__global__ void globaladd(){

}
```

- __device__ : device 에서만 돌아가는 함수
  - Device에서만 호출할 수 있음
- __host__ : host 에서만 돌아가는 함수
- __device__ __host__ : device와 host 두곳 모두 돌아가는 함수
- __global__ : device에서 돌아가는 함수
  - Host에서는 호출할 수 있으나, device에서는 호출할 수 없음

CUDA 간단한 예제

```cpp
#include <cstdio>
#include "device_launch_parameters.h"
#include <cuda_runtime.h>
#include <cstdlib>

__global__ void add(int a, int b, int *c){
    *c = a + b;
}

int main(){
    int c;
    int *dev_c;

    cudaMalloc((void**)&dev_c, sizeof(int));
    add<<< gridDim: 1, blockDim: 1>>>( a: 2, b: 7, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), kind: cudaMemcpyDeviceToHost);

    printf( format: "2+7=%d\n", c);

    cudaFree(dev_c);

    return 0;
}
```

- Malloc() : host 메모리를 할당
- cudaMalloc() : device 메모리를 할당

- cudaMemcpy() : DeviceToHost / HostToDevice 메모리를 복사한다.

- cudaFree() : 할당된 메모리를 해제

CUDA 간단한 예제

```
int main(){
    int *a, *b, *c;
    int *d_a, *d_b, *d_c;

    a = (int *)malloc( size: SIZE*sizeof(int));
    b = (int *)malloc( size: SIZE*sizeof(int));
    c = (int *)malloc( size: SIZE*sizeof(int));

    cudaMalloc(&d_a,  size: SIZE*sizeof(int));
    cudaMalloc(&d_b,  size: SIZE*sizeof(int));
    cudaMalloc(&d_c,  size: SIZE*sizeof(int));

    for(int i=0; i<SIZE; i++){
        a[i] = i; b[i] = i; c[i] = 0;
    }
```

```
    cudaMemcpy(d_a, a,  count: SIZE*sizeof(int),  kind: cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b,  count: SIZE*sizeof(int),  kind: cudaMemcpyHostToDevice);
    cudaMemcpy(d_c, c,  count: SIZE*sizeof(int),  kind: cudaMemcpyHostToDevice);

    VectorAdd<<< gridDim: 1, SIZE>>>(d_a, d_b, d_c, SIZE);

    cudaMemcpy(a, d_a,  count: SIZE*sizeof(int),  kind: cudaMemcpyDeviceToHost);
    cudaMemcpy(b, d_b,  count: SIZE*sizeof(int),  kind: cudaMemcpyDeviceToHost);
    cudaMemcpy(c, d_c,  count: SIZE*sizeof(int),  kind: cudaMemcpyDeviceToHost);

    for(int i=0; i<SIZE; i++){
        printf( format: "c[%d] = %d\n", i, c[i]);
    }

    free(a);
    free(b);
    free(c);

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    return 0;
}
```

CUDA 간단한 예제

```
__global__ void VectorAdd(int *a, int *b, int *c, int n){
    int i = threadIdx.x;

    printf("threadIdx.x : %d, n : %d\n", i, n);

    for(i=0; i<n; i++){
        c[i] = a[i] + b[i];
        printf("%d = %d + %d\n", c[i], a[i], b[i]);
    }
}
```

```
__global__ void VectorAdd(int *a, int *b, int *c, int n){
    int i = threadIdx.x;

    printf("threadIdx.x : %d, n : %d\n", i, n);

    int index = threadIdx.x + blockIdx.x * blockDim.x;
    c[index] = a[index] + b[index];

}
```

```
threadIdx.x : 279, n : 1024
threadIdx.x : 280, n : 1024
threadIdx.x : 281, n : 1024
threadIdx.x : 282, n : 1024
threadIdx.x : 283, n : 1024
threadIdx.x : 284, n : 1024
threadIdx.x : 285, n : 1024
threadIdx.x : 286, n : 1024
threadIdx.x : 287, n : 1024
threadIdx.x : 0, n : 1024
threadIdx.x : 1, n : 1024
threadIdx.x : 2, n : 1024
threadIdx.x : 3, n : 1024
threadIdx.x : 4, n : 1024
threadIdx.x : 5, n : 1024
threadIdx.x : 6, n : 1024
threadIdx.x : 7, n : 1024
threadIdx.x : 8, n : 1024
threadIdx.x : 9, n : 1024
```

```
2046 = 1023 + 1023
2046 = 1023 + 1023
2046 = 1023 + 1023
2046 = 1023 + 1023
2046 = 1023 + 1023
2046 = 1023 + 1023
2046 = 1023 + 1023
2046 = 1023 + 1023
c[0] = 0
c[1] = 2
c[2] = 4
c[3] = 6
c[4] = 8
c[5] = 10
c[6] = 12
c[7] = 14
c[8] = 16
c[9] = 18
c[10] = 20
c[11] = 22
c[12] = 24
c[13] = 26
c[14] = 28
c[15] = 30
c[16] = 32
```

Q & A