1. Hello World

Hello World

Terminology

- host: CPU와 그 메모리(host memory)
  - 호스트는 작업의 전체적인 진행(main 함수)을 제어한다.
- device: GPU와 그 메모리(device memory)
  - 디바이스는 반복적인 작업의 진행(parallelized task)을 제어한다.
  - 하나의 호스트에는 여러개의 디바이스가 붙을 수 있다.

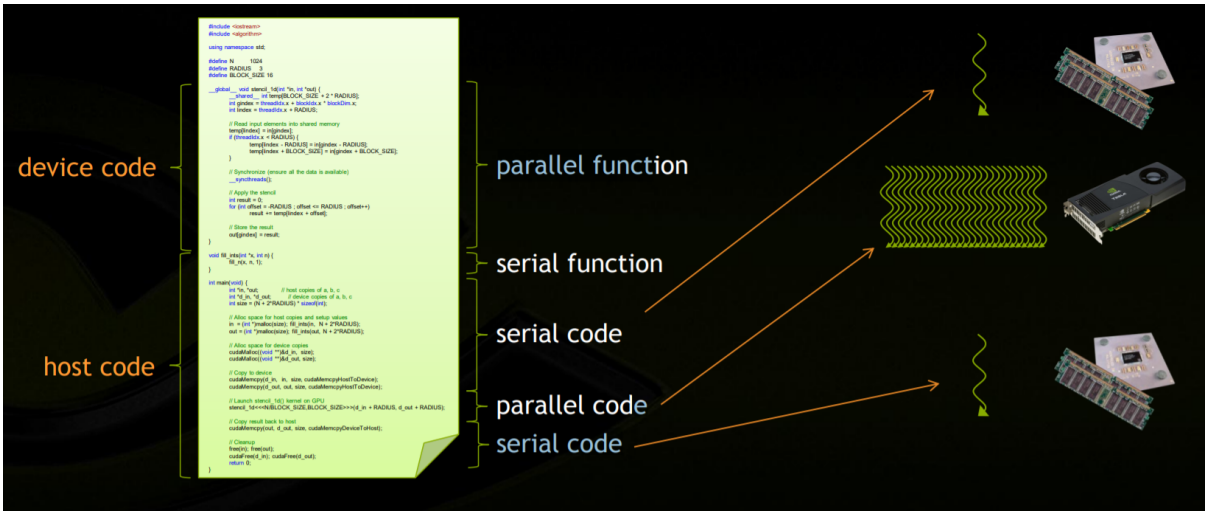
Heterogeneous Computing

- Heterogeneous: 이(二)종의
- Heterogeneous Computing: MPI 코드와 유사하게 하나의 소스 코드에 디바이스와 호스트의 작업(task)을 전부 작성해둔 것.

```
#include <iostream>
#include <algorithm>

using namespace std;

#define N 1024
#define RADIUS 3
#define BLOCK_SIZE 16

__global__ void stencil_1d(int *in, int *out) {
    __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
    int gindex = threadIdx.x + blockIdx.x * blockDim.x;
    int lindex = threadIdx.x + RADIUS;
    // Read input elements into shared memory

    temp[lindex] = in[gindex];

    if (threadIdx.x < RADIUS) {
        temp[lindex - RADIUS] = in[gindex - RADIUS];
        temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
    }
    // Synchronize (ensure all the data is available)

    __syncthreads();
    // Apply the stencil

    int result = 0;
    for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
        result += temp[lindex + offset];
```

```
    // Store the result
    out[gindex] = result;
}
// device function


void fill_ints(int *x, int n) {
    fill_n(x, n, 1);
}
int main(void) {
    int *in, *out; // host copies of a, b, c
    int *d_in, *d_out; // device copies of a, b, c
    int size = (N + 2*RADIUS) * sizeof(int);
    in = (int *)malloc(size); fill_ints(in, N + 2*RADIUS);
    out = (int *)malloc(size); fill_ints(out, N + 2*RADIUS);

    cudaMalloc((void **)&d_in, size);
    cudaMalloc((void **)&d_out, size);

    cudaMemcpy(d_in, in, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_out, out, size, cudaMemcpyHostToDevice);

    stencil_1d<<<N/BLOCK_SIZE,BLOCK_SIZE>>>(d_in + RADIUS, d_out + RADIUS);

    cudaMemcpy(out, d_out, size, cudaMemcpyDeviceToHost);

    free(in); free(out);
    cudaFree(d_in); cudaFree(d_out);

    return 0;
}
// host function
```

- sterncil_1d 함수는 device의 작업을, main 함수는 host의 작업을 작성한 것.
- 보다 자세한 코드 분석은 아래의 그림을 참조할 것.



Processing Flow

1. Copy data from CPU memory to GPU memory
2. Load GPU code and execute it, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

Hello World

Compile and Run

```
#include <iostream>

using namespace std;

int main(void){
    printf("Hello World!\n");

    return 0;
}
```

- nvcc: compile function
  - separates source code into device and host components
    - host: compile in standard host compiler(gcc, cl...)
    - device: compile in NVIDIA compiler
  - Compile and run this program for example

Device Function

```
#include <iostream>

using namespace std;
__global__ void mykernel(void){

}
int main(void){
    mykernel<<<1,1>>>();

    printf("Hello World!\n");

    return 0;
}
```

- `__global__`: means runs on the device(device components). those function most called form host code(main thread)
- `mykernel<<<1,1>>>()`: calling device code
  - Also called a "kernel launch"
  - return to the parameters (1,1)

Example

- add

```
#include <iostream>

using namespace std;
__global__ void add(int *a, int *b, int *c){
    *c = *a + *b;
}
int main(void){
    int a, b, c;
    int *d_a, *d_b, *d_c;
    int size = sizeof(int);

    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    a = 2;
    b = 7;

    cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);

    add<<<1,1>>>(d_a, d_b, d_c);

    cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);

    printf("result: %d", c);

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

    return 0;
}
```

Task to do

1. nvcc로 컴파일된 파일을 어떻게 실행시키는지 찾아보자
2. Heterogeneous Computing 부분에서 예시로 사용된 코드에서 각각 어떤 부분이 어떤 역할을 하는지 알아보고, Processing Flow의 1, 2, 3이 각각 어떤 부분에 해당되는지 알아보자.