Basic framework of simple CUDA programs

For a simple CUDA program written in a single source file, the basic framework is as follows:

```
header inclusion
const or macro definition
declarations of C++ functions and CUDA kernels
int main()
{
    allocate host and device memory
    initialize data in host memory
    transfer data from host to device
    launch (call) kernel to do calculations in the device
    transfer data from device to host
    free host and device memory
}
definitions of C++ functions and CUDA kernels
```

In our CUDA program, first we defined pointers, as follows

```
double *d_x, *d_y, *d_z;
```

and used the cudaMalloc() function to allocate memory in device.

```
cudaError_t cudaMalloc(void **address, size_t size);
```

Here, address is the address of the pointer (so it is a double pointer), size is the number of bytes to be allocated,
and cudaSuccess is a return value indicating whether there is error when calling this function.

```
    cudaMalloc((void **)&d_x, M);
    cudaMalloc((void **)&d_y, M);
    cudaMalloc((void **)&d_z, M);
```

M is sizeof(double) * N, where N is the number of elements in an array, and sizeof(double) is the memory size (number of bytes) for a double-precision floating point number. The type conversion (void

**) can be omitted, i.e., we can change the above lines to:

```
    cudaMalloc(&d_x, M);
    cudaMalloc(&d_y, M);
    cudaMalloc(&d_z, M);
```

Memory allocated by cudaMalloc() needs to be freed by using the cudaFree() function:

```
cudaError_t cudaFree(void* address);
```

Data transfer between host and device

We can transfer (copy) some data from host to device after allocating the device memory,

```
cudaError_t cudaMemcpy(
    void                    *dst,
    const void              *src,
    size_t                  count,
    enum cudaMemcpyKind kind);
```

Here, dst is the address of the destination (to be transferred to), src is the address of the source (to be transferred from), count is the number of bytes to transferred , and kind indicates the direction of the data transfer. The possible values of the enum parameter kind include cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, cudaMemcpyDeviceToDevice, and cudaMemcpyDefault. The meanings of the first 4 are obvious and for the last one, it means that transfer direction will be automatically inferred from the pointers dst and src. This automatic process requires that the host system is 64 bit supporting unified virtual addressing.

Some requirements for kernels

- A kernel must return void.
- A kernel must be decorated by __global__.

The necessity of if statements in most kernels

In general, when the number of elements cannot be divided by the block size, the grid size can be calculated in one of the following ways:

They are both equivalent to the following statement:

```
int grid_size = (N % block_size == 0)
                      ? (N / block_size)
                      : (N / block_size + 1);
```

Because now the number of threads (10^8+128) exceeds the number of elements (10^8+1), we must use an if statement to avoid manipulating invalid addresses:

```
    const int n = blockDim.x * blockIdx.x + threadIdx.x;
    if (n < N)
    {
        z[n] = x[n] + y[n];
    }
```