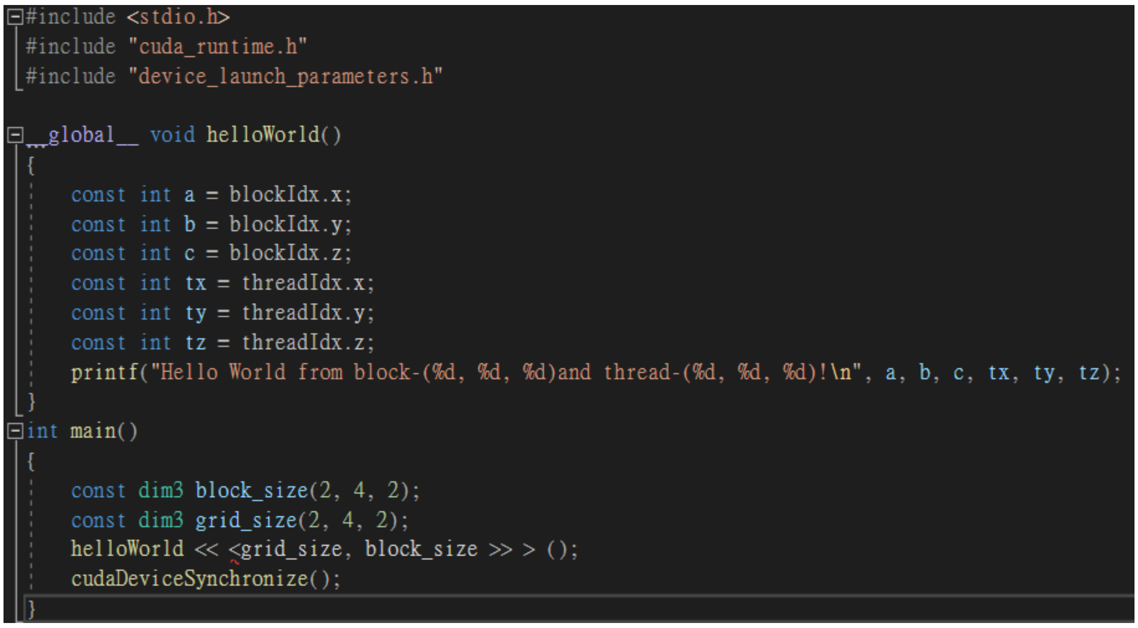
# Thread-organization



```
#include <stdio.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

__global__ void helloWorld()
{
    const int a = blockIdx.x;
    const int b = blockIdx.y;
    const int c = blockIdx.z;
    const int tx = threadIdx.x;
    const int ty = threadIdx.y;
    const int tz = threadIdx.z;
    printf("Hello World from block-(%d, %d, %d)and thread-(%d, %d, %d)!\n", a, b, c, tx, ty, tz);
}
int main()
{
    const dim3 block_size(2, 4, 2);
    const dim3 grid_size(2, 4, 2);
    helloWorld << <grid_size, block_size >> > ();
    cudaDeviceSynchronize();
}
```

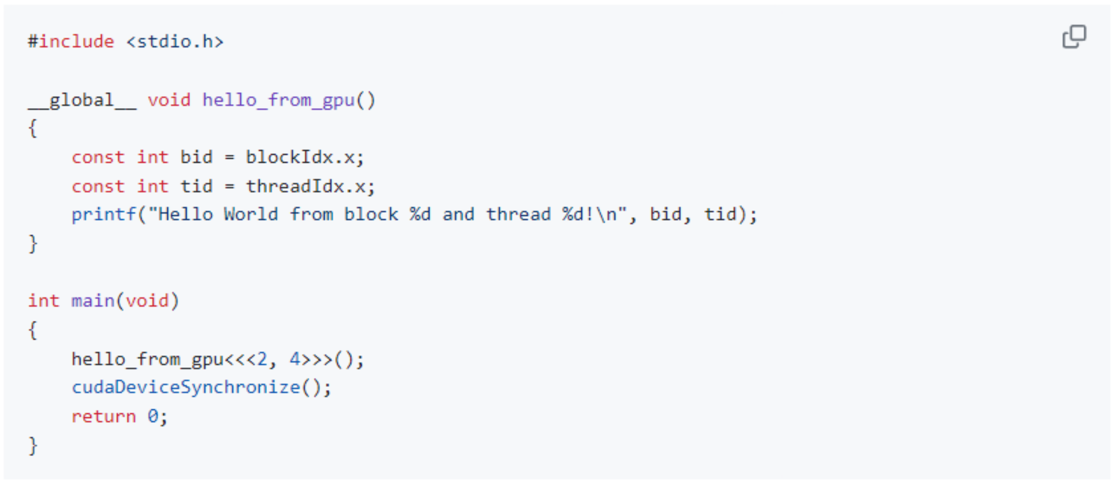
Use the construct dim3 to define multi- dimensional grids and blocks:

```
    dim3 grid-size(Gx,Gy,Gz);
    dim3 block-size(Bx,By,Bz);
```

If the size of z dimension is 1, we can simplify the above definitions to:

```
    dim3 grid-size(Gx,Gy);
    dim3 block-size(Bx,By);
```

To demonstrate the usage of a multi-dimensional block, we write our last version of the Hello World program:



```
#include <stdio.h>

__global__ void hello_from_gpu()
{
    const int bid = blockIdx.x;
    const int tid = threadIdx.x;
    printf("Hello World from block %d and thread %d!\n", bid, tid);
}

int main(void)
{
    hello_from_gpu<<<2, 4>>>();
    cudaDeviceSynchronize();
    return 0;
}
```

The output of this program is:

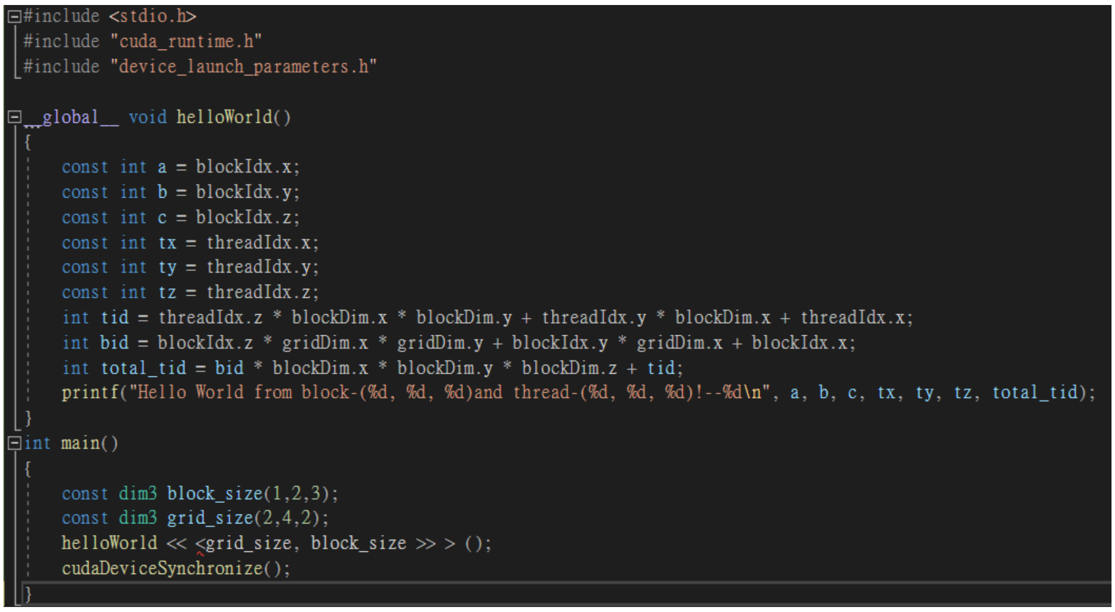
```
Hello World from block-0 and thread-(0, 0)!
Hello World from block-0 and thread-(1, 0)!
Hello World from block-0 and thread-(0, 1)!
Hello World from block-0 and thread-(1, 1)!
Hello World from block-0 and thread-(0, 2)!
Hello World from block-0 and thread-(1, 2)!
Hello World from block-0 and thread-(0, 3)!
Hello World from block-0 and thread-(1, 3)!
```

If we label the lines as 0-7 from top to down, this label can be calculated as
`threadIdx.y * blockDim.x + threadIdx.x = threadIdx.y * 2 + threadIdx.x`:

```
Hello World from block-0 and thread-(0, 0)! // 0 = 0 * 2 + 0
Hello World from block-0 and thread-(1, 0)! // 1 = 0 * 2 + 1
Hello World from block-0 and thread-(0, 1)! // 2 = 1 * 2 + 0
Hello World from block-0 and thread-(1, 1)! // 3 = 1 * 2 + 1
Hello World from block-0 and thread-(0, 2)! // 4 = 2 * 2 + 0
Hello World from block-0 and thread-(1, 2)! // 5 = 2 * 2 + 1
Hello World from block-0 and thread-(0, 3)! // 6 = 3 * 2 + 0
Hello World from block-0 and thread-(1, 3)! // 7 = 3 * 2 + 1
```

In general, the one-dimensional index tid of a thread is related to the multi-dimensional indices of the thread via the the following relation:

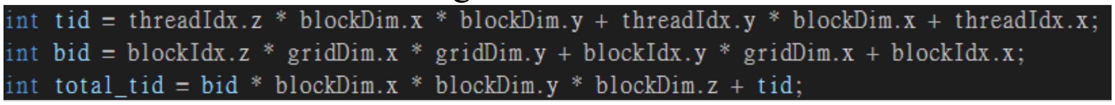
```
int tid = threadIdx.z * blockDim.x * blockDim.y + threadIdx.y * blockDim.x + threadIdx.x;
```



```
#include <stdio.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

__global__ void helloWorld()
{
    const int a = blockIdx.x;
    const int b = blockIdx.y;
    const int c = blockIdx.z;
    const int tx = threadIdx.x;
    const int ty = threadIdx.y;
    const int tz = threadIdx.z;
    int tid = threadIdx.z * blockDim.x * blockDim.y + threadIdx.y * blockDim.x + threadIdx.x;
    int bid = blockIdx.z * gridDim.x * gridDim.y + blockIdx.y * gridDim.x + blockIdx.x;
    int total_tid = bid * blockDim.x * blockDim.y * blockDim.z + tid;
    printf("Hello World from block-(%d, %d, %d)and thread-(%d, %d, %d)!--%d\n", a, b, c, tx, ty, tz, total_tid);
}
int main()
{
    const dim3 block_size(1,2,3);
    const dim3 grid_size(2,4,2);
    helloWorld << <grid_size, block_size >> > ();
    cudaDeviceSynchronize();
}
```

In multi- dimensional case , If we need thread-id in whole cuda kernel, the thread via the the following relation:



```
int tid = threadIdx.z * blockDim.x * blockDim.y + threadIdx.y * blockDim.x + threadIdx.x;
int bid = blockIdx.z * gridDim.x * gridDim.y + blockIdx.y * gridDim.x + blockIdx.x;
int total_tid = bid * blockDim.x * blockDim.y * blockDim.z + tid;
```

Limits on the grid and block sizes:

For all the GPUs starting from the Kepler architecture, the grid size is limited to:

```
gridDim.x <= 2^{31}-1
gridDim.y <= 2^{16}-1 = 65535
gridDim.z <= 2^{16}-1 = 65535
```

and the block size is limited to:

```
blockDim.x <= 1024
blockDim.y <= 1024
blockDim.z <= 64
```

Besides this, there is an important limit on the following product:

```
blockDim.x * blockDim.y * blockDim.z <= 1024
```

## It is important to remember the above limits.