Lab9 – CUDA environment

Compile a CUDA program.

```
nvcc –o prog prog.cu // ignore the warning error
```

```
Login ID csxx(xx:even): cudaSetDevice(0);
                (xx:odd): cudaSetDevice(1);
```

ex.1 Check CUDA 1-d indices inside a CUDA kernel.

```
#include <stdio.h>

#define X 12
#define THREAD_X 4

__global__ void index(int *A)
{
    int i = blockDim.x*blockIdx.x+threadIdx.x;

    (1) A[i] = 123;
    (2) A[i] = i;
    (3) A[i] = gridDim.x;
    (4) A[i] = blockDim.x;
    (5) A[i] = threadIdx.x;
}

int main()
{
    int A[X], *A_d, i;

    dim3 dimBlock(THREAD_X);
    dim3 dimGrid(X/dimBlock.x);

    cudaSetDevice(0); // or 1
    cudaMalloc((void **)&A_d, sizeof(int)*X);

    for (i=0; i<X; i++)
        A[i] = -1;

    cudaMemcpy(A_d, A, sizeof(int)*X, cudaMemcpyHostToDevice);

    index<<<dimGrid, dimBlock>>>(A_d);

    cudaMemcpy(A, A_d, sizeof(int)*X, cudaMemcpyDeviceToHost);

    for (i=0; i<X; i++)
        printf("%d ", A[i]);
    printf("₩n");

    cudaFree(A_d);
}
```

ex.2 Check CUDA 2-d indices inside a CUDA kernel.

```
#include <stdio.h>

#define X 9
#define Y 8

#define THREAD_X 3
#define THREAD_Y 2

#define A(i,j) A[i*Y+j]

__global__ void index(int *A)
{
    int i = blockDim.x*blockIdx.x+threadIdx.x;
    int j = blockDim.y*blockIdx.y+threadIdx.y;

    (1)  A(i,j) = threadIdx.x;
    (2)  A(i,j) = threadIdx.y;
    (3)  A(i,j) = blockIdx.y;
    (4)  A(i,j) = blockIdx.y;
    (5)  A(i,j) = gridDim.x;
    (6)  A(i,j) = gridDim.y;
    (7)  A(i,j) = blockDim.x;
    (8)  A(i,j) = blockDim.y;
    (9)  A(i,j) = i;
    (10)A(i,j) = j;
    (11)A(i,j) = i*Y+j;
}
int main()
{
    int A[X][Y], *A_d;
    int i, j;

    dim3 dimBlock(THREAD_X,THREAD_Y);
    dim3 dimGrid(X/dimBlock.x,Y/dimBlock.y);
    cudaSetDevice(0); // or 1

    cudaMalloc((void **)&A_d, sizeof(int)*X*Y);

    for (i=0; i<X; i++)
        for (j=0; j<Y; j++)
            A[i][j] = -1;

    cudaMemcpy(A_d, A, sizeof(int)*X*Y, cudaMemcpyHostToDevice);

    index<<<dimGrid, dimBlock>>>(A_d);

    cudaMemcpy(A, A_d, sizeof(int)*X*Y, cudaMemcpyDeviceToHost);

    for (i=0; i<X; i++) {
        for (j=0; j<Y; j++)
            printf("%d ", A[i][j]);
        printf("₩n");
    }

    cudaFree(A_d);
}
```

ex.3 Complete the following CUDA program(gradual.cu).

```
#include <stdio.h>

#define N 12

#define A(i,j) A[i*N+j]

__global__ void gradual(int *A)
{
    // COMPLETE THIS AREA
}

int main(int argc, char *argv[])
{
    int A[N][N], *A_d;
    int nt, i, j, dev;

    // GPU info
    cudaDeviceProp deviceProp;

    dev = 1;
    cudaSetDevice(dev);
    cudaGetDevice(&dev);
    cudaGetDeviceProperties(&deviceProp, dev);
    printf("Using Device %d: ₩"%s₩"₩n", dev, deviceProp.name);

    if (argc != 2) {
        fprintf(stderr, "usage: %s #threads₩n", argv[0]);
        exit(1);
    }
    nt = atoi(argv[1]);

    dim3 dimBlock(...);
    dim3 dimGrid(...);

    cudaMalloc(...);

    gradual<<<...>>>(A_d);

    cudaMemcpy(...);

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++)
            printf("%3d ", A[i][j]);
        printf("₩n");
    }

    cudaFree(A_d);
}
```

Submit the program when you are done.

Run only 3x3, 4x4, 6x6, 12x12 threads. The output is: