Started on	Thursday, September 8, 2022, 3:09 PM
State	Finished
Completed on	Thursday, September 8, 2022, 3:39 PM
Time taken	30 mins 43 secs
Grade	**19.00** out of 21.00 (**90**%)

Question **1**

Complete

1.00 points out of 1.00

If we want to allocate an array of v integer elements in CUDA device global memory, what would be an appropriate expression for the second argument of the cudaMalloc() call?

Select one:

- a. n
- b. v
- c. n * sizeof(int)
- d. v * sizeof(int)
- e. none of the answers

Question **2**

Complete

1.00 points out of 1.00

If we want to allocate an array of n floating-point elements and have a floating-point pointer variable d_A to point to the allocated memory, what would be an appropriate expression for the first argument of the cudaMalloc() call?

Select one:

- a. n
- b. (void *) d_A
- c. *d_A
- d. (void **) & d_A
- e. none of the answers

Question **3**

Complete

1.00 points out of 1.00

If we want to copy 3000 bytes of data from host array h_A (h_A is a pointer to element 0 of the source array) to device array d_A (d_A is a pointer to element 0 of the destination array), what would be an appropriate API call for this in CUDA?

Select one:

- a. cudaMemcpy(3000, h_A, d_A, cudaMemcpyHostToDevice);
- b. cudaMemcpy(h_A, d_A, 3000, cudaMemcpyDeviceToHost);
- c. cudaMemcpy(d_A, h_A, 3000, cudaMemcpyHostToDevice);
- d. cudaMemcpy(3000, d_A, h_A, cudaMemcpyHostToDevice);

Question **4**

Complete

1.00 points out of 1.00

How would one declare a variable err that can appropriately receive returned value of a CUDA API call?

Select one:

- a. int err;
- b. cudaError err;
- c. cudaError_t err;
- d. cudaSuccess_t err;

Question **5**

Complete

5.00 points out of 5.00

The following CUDA C code implements summation with a 2D grid that contains 2D blocks:

```
#include "../common/common.h"
#include <cuda_runtime.h>
#include <stdio.h>

/*
 * This example demonstrates a simple vector sum
 * on the GPU and on the host.
 * sumArraysOnGPU splits the work of the vector sum
 * across CUDA threads on the GPU.
 * A 2D thread block and 2D grid are used.
 * sumMatrixOnHost sequentially
 * iterates through vector elements on the host.
 */

void initialData(float *ip, const int size)
{
    int i;

    for(i = 0; i < size; i++)
    {
        ip[i] = (float)(rand() & 0xFF) / 10.0f;
    }

    return;
}

void sumMatrixOnHost(float *A, float *B, float *C, const int nx,
                     const int ny)
{
    float *ia = A;
    float *ib = B;
    float *ic = C;

    for (int iy = 0; iy < ny; iy++)
    {
        for (int ix = 0; ix < nx; ix++)
        {
            ic[ix] = ia[ix] + ib[ix];

        }

        ia += nx;
        ib += nx;
        ic += nx;
    }

    return;
}
```

```
}


void checkResult(float *hostRef, float *gpuRef, const int N)
{
    double epsilon = 1.0E-8;
    bool match = 1;

    for (int i = 0; i < N; i++)
    {
        if (abs(hostRef[i] - gpuRef[i]) > epsilon)
        {
            match = 0;
            printf("host %f gpu %f\n", hostRef[i], gpuRef[i]);
            break;
        }
    }

    if (match)
        printf("Arrays match.\n\n");
    else
        printf("Arrays do not match.\n\n");
}

// grid 2D block 2D
__global__ void sumMatrixOnGPU2D(float *MatA, float *MatB,
                                 float *MatC, int nx,
                                 int ny)
{
        //@Complete the code below (1)
}

int main(int argc, char **argv)
{
    printf("%s Starting...\n", argv[0]);

    // set up device
    int dev = 0;
    cudaDeviceProp deviceProp;
    CHECK(cudaGetDeviceProperties(&deviceProp, dev));
    printf("Using Device %d: %s\n", dev, deviceProp.name);
    CHECK(cudaSetDevice(dev));

    // set up data size of matrix = 2^28
    int nx = 1 << 14;
    int ny = 1 << 14;

    int nxy = nx * ny;
    int nBytes = nxy * sizeof(float);
    printf("Matrix size: nx %d ny %d\n", nx, ny);
```

```
// malloc host memory

float *h_A, *h_B, *hostRef, *gpuRef;
h_A = (float *)malloc(nBytes);
h_B = (float *)malloc(nBytes);
hostRef = (float *)malloc(nBytes);
gpuRef = (float *)malloc(nBytes);

// initialize data at host side
double iStart = seconds();
initialData(h_A, nxy);
initialData(h_B, nxy);
double iElaps = seconds() - iStart;
printf("Matrix initialization elapsed %f sec\n", iElaps);

memset(hostRef, 0, nBytes);
memset(gpuRef, 0, nBytes);

// add matrix at host side for result checks
iStart = seconds();
sumMatrixOnHost(h_A, h_B, hostRef, nx, ny);
iElaps = seconds() - iStart;
printf("sumMatrixOnHost elapsed %f sec\n", iElaps);

// malloc device global memory
float *d_MatA, *d_MatB, *d_MatC;
//@Complete the code below - (2)

// transfer data from host to device
//@Complete the code below - (3)

// invoke kernel at host side;
int dimx = 32;
int dimy = 32;
// grid/block configuration
//@Complete the code below - (4)

iStart = seconds();
//call the kernel function
//@Complete the code below - (5)

//@Complete the code below - (6)


iElaps = seconds() - iStart;
printf("sumMatrixOnGPU2D <<<(%d,%d),
                (%d,%d)>>> elapsed %f sec\n", grid.x,
      grid.y,
      block.x, block.y, iElaps);
// check kernel error
CHECK(cudaGetLastError());
```

// copy kernel result back to host side
//@Complete the code below - (7)

// check device results
checkResult(hostRef, gpuRef, nxy);

// free device global memory
//@Complete the code below - (8)

// free host memory
free(h_A);
free(h_B);
free(hostRef);
free(gpuRef);

// reset device
CHECK(cudaDeviceReset());

return (0);
}

where common.h is defined as follows:

```
#include <sys/time.h>

#ifndef _COMMON_H
#define _COMMON_H

#define CHECK(call)                                             \
{                                                               \
    const cudaError_t error = call;                             \
    if (error != cudaSuccess)                                   \
    {                                                           \
        fprintf(stderr, "Error: %s:%d, ", __FILE__, __LINE__);\
        fprintf(stderr, "code: %d, reason: %s\n", error,      \
                cudaGetErrorString(error));                     \
        exit(1);                                                \
    }                                                           \
}

inline double seconds()
{
    struct timeval tp;
    struct timezone tzp;
    int i = gettimeofday(&tp, &tzp);
    return ((double)tp.tv_sec + (double)tp.tv_usec * 1.e-6);
}

#endif // _COMMON_H
```

Complete the code for (1).

int x = blockIdx.x * blockDim.x + threadIdx.x;
int y = blockIdx.y * blockDim.y + threadIdx.y;
int matID = y * nx + x;

if (x < nx && y < ny)
{

```
    MatC[matID] = MatA[matID] + MatB[matID];

}
```

Question **6**

Complete

3.00 points out of 3.00

Complete the code for (2).

```
cudaMalloc( (void**)&d_MatA, nBytes);
cudaMalloc( (void**)&d_MatB, nBytes);
cudaMalloc( (void**)&d_MatC, nBytes);
```

Question **7**

Complete

2.00 points out of 2.00

Complete the code for (3).

```
cudaMemcpy( d_MatA, h_A, nBytes, cudaMemcpyHostToDevice);
cudaMemcpy( d_MatB, h_B, nBytes, cudaMemcpyHostToDevice);
```

Question **8**

Complete

1.00 points out of 2.00

Complete the code for (4).

```
dim3 grid { std::ceil(nx / 32), std::ceil(nx / 32), 1};
dim3 block { nx, ny, 1};
```

Question **9**

Complete

1.00 points out of 1.00

Complete the code for (5).

```
sumMatrixOnGPU2D<<< grid, block >>> (d_MatA, d_MatB, d_MatC, nx, ny);
```

Question **10**

Complete

0.00 points out of 1.00

Complete the code for (6).

```
sumMatrixOnGPU2D<<< grid, block >>> (d_MatA, d_MatB, d_MatC, nx, ny);
```

Question **11**

Complete

Not graded

Complete the code for (7).

```
cudaMemcpy( gpuRef , d_MatC, nBytes, cudaMemcpyDeviceToHost);

//Check function uses gpuref instead of h_C
```

Question **12**

Complete

3.00 points out of 3.00

Complete the code for (8).

```
cudaFree(d_MatA);
cudaFree(d_MatB);
cudaFree(d_MatC);
```