CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#define BLOCK_SIZE 16
__global__ void vectorAdd(int *A, int *B, int *C, int n) {
   int i = threadIdx.x + blockIdx.x * blockDim.x;
   if (i < n) {
      C[i] = A[i] + B[i];
   }
}
__global__ void matrixMultiply(int *A, int *B, int *C, int N) {
   int row = blockIdx.y * blockDim.y + threadIdx.y;
   int col = blockIdx.x * blockDim.x + threadIdx.x;
   if (row < N && col < N) {
      int sum = 0;
      for (int k = 0; k < N; k++) {
         sum += A[row * N + k] * B[k * N + col];
      }
      C[row * N + col] = sum;
   }
}
void vectorAddition() {
   int n;
   printf("Enter the size of the vectors: ");
   scanf("%d", &n);
   int *h_A, *h_B, *h_C;
   int *d_A, *d_B, *d_C;
   size_t size = n * sizeof(int);
   h_A = (int *)malloc(size);
   h_B = (int *)malloc(size);
   h_C = (int *)malloc(size);
   printf("Enter elements of first vector: \n");
   for (int i = 0; i < n; i++) scanf("%d", &h_A[i]);
   printf("Enter elements of second vector: \n");
   for (int i = 0; i < n; i++) scanf("%d", &h_B[i]);
   cudaMalloc((void **)&d_A, size);
   cudaMalloc((void **)&d_B, size);
   cudaMalloc((void **)&d_C, size);
   cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
```

```
    int threadsPerBlock = 256;
    int blocksPerGrid = (n + threadsPerBlock - 1) / threadsPerBlock;
    vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, n);
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    printf("Resultant Vector: \n");
    for (int i = 0; i < n; i++) printf("%d ", h_C[i]);
    printf("\n");
    free(h_A); free(h_B); free(h_C);
    cudaFree(d_A); cudaFree(d_B); cudaFree(d_C);
}
void matrixMultiplication() {
    int N;
    printf("Enter the size of the NxN matrix: ");
    scanf("%d", &N);
    int *h_A, *h_B, *h_C;
    int *d_A, *d_B, *d_C;
    size_t size = N * N * sizeof(int);
    h_A = (int *)malloc(size);
    h_B = (int *)malloc(size);
    h_C = (int *)malloc(size);
    printf("Enter elements of first matrix: \n");
    for (int i = 0; i < N * N; i++) scanf("%d", &h_A[i]);
    printf("Enter elements of second matrix: \n");
    for (int i = 0; i < N * N; i++) scanf("%d", &h_B[i]);
    cudaMalloc((void **)&d_A, size);
    cudaMalloc((void **)&d_B, size);
    cudaMalloc((void **)&d_C, size);
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
    dim3 block(BLOCK_SIZE, BLOCK_SIZE);
    dim3 grid((N + BLOCK_SIZE - 1) / BLOCK_SIZE, (N + BLOCK_SIZE - 1) / BLOCK_SIZE);
    matrixMultiply<<<grid, block>>>(d_A, d_B, d_C, N);
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    printf("Resultant Matrix: \n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", h_C[i * N + j]);
        }
        printf("\n");
    }
    free(h_A); free(h_B); free(h_C);
    cudaFree(d_A); cudaFree(d_B); cudaFree(d_C);
}
int main() {
    int choice;
    printf("Choose an operation:\n");
    printf("1. Vector Addition\n");
    printf("2. Matrix Multiplication\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    if (choice == 1) {
        vectorAddition();
    } else if (choice == 2) {
        matrixMultiplication();
    } else {
        printf("Invalid choice!\n");
    }
    return 0;
}
```

INPUT:

```
Choose an operation:
1. Vector Addition
2. Matrix Multiplication
Enter your choice: 1
Enter the size of the vectors: 5
Enter elements of first vector:
1 2 3 4 5
Enter elements of second vector:
5 4 3 2 1
```

OUTPUT:

```
Resultant Vector:
6 6 6 6 6
```

INPUT:

```
Choose an operation:
1. Vector Addition
2. Matrix Multiplication
Enter your choice: 2
Enter the size of the NxN matrix: 3
Enter elements of first matrix:
1 2 3
4 5 6
7 8 9
Enter elements of second matrix:
9 8 7
6 5 4
3 2 1
```

OUTPUT:

```
Resultant Matrix:
30 24 18
84 69 54
138 114 90
```