CUDA Program For:

1>Addition Of Two Large Vectors

```
#include <stdio.h>

_global_ void add_vectors(int *a, int *b, int *c, int n)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index < n)
    {
        c[index] = a[index] + b[index];
    }
}

int main()
{
    int n = 1000000;
    int *a, *b, *c; // host arrays
    int *d_a, *d_b, *d_c; // device arrays
    int size = n * sizeof(int);
    a = (int *)malloc(size);
    b = (int *)malloc(size);
    c = (int *)malloc(size);
    for (int i = 0; i < n; i++)
    {
        a[i] = i;
        b[i] = 2 * i;
    }
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);
    cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);
    int threads_per_block = 256;
    int blocks_per_grid = (n + threads_per_block - 1) / threads_per_block;
    add_vectors<<<blocks_per_grid, threads_per_block>>>(d_a, d_b, d_c, n);
    cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    for (int i = 0; i < n; i++)
    {
        printf("%d ", c[i]);
    }
    free(a);
    free(b);
    free(c);

    return 0;
}
```

2>Matrix Multiplication Using CUDA C

```
#include <stdio.h>

#define N 1024 // size of matrices
#define THREADS_PER_BLOCK 32

_global_ void matrixMultiply(float *a, float *b, float *c, int n)
```

```
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    float sum = 0;
    for (int i = 0; i < n; i++) {
        sum += a[row * n + i] * b[i * n + col];
    }

    c[row * n + col] = sum;
}

int main()
{
    float *a, *b, *c; // matrices
    float *d_a, *d_b, *d_c;
    a = (float *) malloc(N * N * sizeof(float));
    b = (float *) malloc(N * N * sizeof(float));
    c = (float *) malloc(N * N * sizeof(float));
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            a[i * N + j] = i * N + j;
            b[i * N + j] = j * N + i;
            c[i * N + j] = 0;
        }
    }
    cudaMalloc(&d_a, N * N * sizeof(float));
    cudaMalloc(&d_b, N * N * sizeof(float));
    cudaMalloc(&d_c, N * N * sizeof(float));
    cudaMemcpy(d_a, a, N * N * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, N * N * sizeof(float), cudaMemcpyHostToDevice);
    dim3 gridSize((N + THREADS_PER_BLOCK - 1) / THREADS_PER_BLOCK, (N + THREADS_PER_BLOCK - 1) / THREADS_PER_BLOCK);
    dim3 blockSize(THREADS_PER_BLOCK, THREADS_PER_BLOCK);
    matrixMultiply<<<gridSize, blockSize>>>(d_a, d_b, d_c, N);
    cudaMemcpy(c, d_c, N * N * sizeof(float), cudaMemcpyDeviceToHost);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%f ", c[i * N + j]);
        }
        printf("\n");
    }
    free(a);
    free(b);
    free(c);
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    return 0;
}
```