CUDA

MATRIX MULTIPLICATION

```
#include <stdio.h>

#define N 3

__global__ void matrixMultiplication(float *A, float *B, float *C, int n)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;

    if (i < n && j < n) {
        float sum = 0.0f;
        for (int k = 0; k < n; ++k) {
            sum += A[i * n + k] * B[k * n + j];
        }
        C[i * n + j] = sum;
    }
}

int main()
{
    float A[N][N] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    float B[N][N] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};
    float C[N][N] = {0};

    // Allocate device memory
    float *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, N * N * sizeof(float));
    cudaMalloc(&d_B, N * N * sizeof(float));
    cudaMalloc(&d_C, N * N * sizeof(float));

    // Copy input matrices from host to device
    cudaMemcpy(d_A, A, N * N * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, N * N * sizeof(float), cudaMemcpyHostToDevice);

    // Set the grid and block dimensions
    dim3 gridDim(ceil(N/16.0), ceil(N/16.0), 1);
    dim3 blockDim(16, 16, 1);

    // Launch the kernel
    matrixMultiplication<<<gridDim, blockDim>>>(d_A, d_B, d_C, N);

    // Copy result matrix from device to host
    cudaMemcpy(C, d_C, N * N * sizeof(float), cudaMemcpyDeviceToHost);

    // Print the result matrix
```

```
    printf("Result Matrix:\n");
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            printf("%.1f ", C[i][j]);

        }
        printf("\n");
    }

    // Free device memory
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

```
ubuntu@DESKTOP-HE9T2TD:~/LP5/Assignment4$ nvcc -o mul matrix_multiplication.cu
ubuntu@DESKTOP-HE9T2TD:~/LP5/Assignment4$ ./mul
Result Matrix:
30.0 24.0 18.0
84.0 69.0 54.0
138.0 114.0 90.0
```

VECTOR ADDITION

```
#include <stdio.h>
#include <stdlib.h>

#define N 5

__global__ void add(int *a, int *b, int *c) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < N) {
        c[i] = a[i] + b[i];
    }
}

int main() {
    int a[N] = {1, 2, 3, 4, 5};
    int b[N] = {6, 7, 8, 9, 10};
    int c[N] = {0};

    int *dev_a, *dev_b, *dev_c;

    cudaMalloc((void **)&dev_a, N * sizeof(int));
    cudaMalloc((void **)&dev_b, N * sizeof(int));
    cudaMalloc((void **)&dev_c, N * sizeof(int));

    cudaMemcpy(dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, N * sizeof(int), cudaMemcpyHostToDevice);

    add<<<1, N>>>(dev_a, dev_b, dev_c);

    cudaMemcpy(c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost);

    for (int i = 0; i < N; i++) {
        //printf("%d ", c[i]);
        printf("%d + %d = %d\n", a[i], b[i], c[i]);
    }
    printf("\n");

    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);

    return 0;
}
```

OUTPUT

```
ubuntu@DESKTOP-HE9T2TD:~/LP5/Assignment4$ nvcc -o add vector_addition.cu
ubuntu@DESKTOP-HE9T2TD:~/LP5/Assignment4$ ./add
1 + 6 = 7
2 + 7 = 9
3 + 8 = 11
4 + 9 = 13
5 + 10 = 15
```