PRATAP SHINGANE
2020BTEIT00050
COURSE: PC

1C) Matrix Addition CUDA code

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

#define N 10

__global__ void MatAdd(int* A, int* B, int* C, int n){
    // Calculate the element index of the current thread
    int i = threadIdx.x;
    int j = threadIdx.y;

    if (i < n && j < n) {
        // Calculate the linear index of the element in 1D array
        int idx = i * n + j;
        // Add the corresponding elements of A and B and store the result in C
        C[idx] = A[idx] + B[idx];
    }
}

void initialize_matrix(int* mat, int n, int val){
    // Initialize a square matrix of size n with a constant value
    for (int i = 0; i < n*n; i++) {
        mat[i] = val;
    }
}

void print_matrix(int* mat, int n){
    // Print a square matrix of size n
    for(int i = 0; i < n; i++){
```

```
        for(int j = 0; j < n; j++){
            printf("%d ", mat[i*n+j]);
        }
        printf("\n");
    }
}

int main(){
    int* A, *B, *C;
    int size = N * N * sizeof(int);

    // Allocate memory on the host for matrices A, B, and C
    A = (int*)malloc(size);
    B = (int*)malloc(size);
    C = (int*)malloc(size);

    // Initialize matrices A and B
    initialize_matrix(A, N, 1);
    initialize_matrix(B, N, 2);

    // Print matrices A and B
    printf("A: \n");
    print_matrix(A, N);
    printf("B: \n");
    print_matrix(B, N);

    // Allocate memory on the device for matrices A, B, and C
    int* d_A, *d_B, *d_C;
    cudaMalloc((void**)&d_A, size);
    cudaMalloc((void**)&d_B, size);
    cudaMalloc((void**)&d_C, size);

    // Copy matrices A and B from host to device
    cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice);
```

```
 // To get the elapsed time
cudaEvent_t start, end;
cudaEventCreate(&start);
cudaEventCreate(&end);

cudaEventRecord(start);


// Launch kernel to perform matrix addition
dim3 threadsPerBlock(N, N);
MatAdd<<<1, threadsPerBlock>>>(d_A, d_B, d_C, N);

 cudaEventRecord(end);
cudaEventSynchronize(end);

float elapsed_time_ms;
cudaEventElapsedTime(&elapsed_time_ms, start, end);


cudaEventDestroy(start);
cudaEventDestroy(end);

// Copy result matrix C from device to host
cudaMemcpy(C, d_C, size, cudaMemcpyDeviceToHost);

// Print result matrix C
printf("C: \n");
print_matrix(C, N);

printf("\n\n\nElapsed time: %f ms\n", elapsed_time_ms);

// Free memory
free(A);
free(B);
```

```
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

OUTPUT: