Modern Education Society's
Wadia College of Engineering, Pune

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: HPC-04**

TITLE: VECTOR AND MATRIX OPERATIONS.

PROBLEM STATEMENT: Write a CUDA Program for:

1. Addition of two large vectors
2. Matrix Multiplication using CUDA

OBJECTIVES:

1. Used of CUDA to perform vector ,matrix operations.
2. To analyze the performance of parallel algorithm.

PRE-REQUISITES:

Knowledge of dividing a large problem into sub-problems. Identify data dependency among them. Knowledge of basic matrix and vector operations.

Steps for Addition of two large vectors using CUDA

1. Define the size of the vectors: In this step, you need to define the size of the vectors that you want to add. This will determine the number of threads and blocks you will need to use to parallelize the addition operation.
2. Allocate memory on the host: In this step, you need to allocate memory on the host for the two vectors that you want to add and for the result vector. You can use the C malloc function to allocate memory.
3. Initialize the vectors: In this step, you need to initialize the two vectors that you want to add on the host. You can use a loop to fill the vectors with data.
4. Allocate memory on the device: In this step, you need to allocate memory on the device for the two vectors that you want to add and for the result vector. You can use the CUDA function cudaMalloc to allocate memory.
5. Copy the input vectors from host to device: In this step, you need to copy the two input vectors from the host to the device memory. You can use the CUDA function cudaMemcpy to copy the vectors.

6. Launch the kernel: In this step, you need to launch the CUDA kernel that will perform the addition operation. The kernel will be executed by multiple threads in parallel. You can use the <<<...>>>
   syntax to specify the number of blocks and threads to use.
7. Copy the result vector from device to host: In this step, you need to copy the result vector from the device memory to the host memory. You can use the CUDA function cudaMemcpy to copy the result vector.
8. Free memory on the device: In this step, you need to free the memory that was allocated on the device. You can use the CUDA function cudaFree to free the memory.
9. Free memory on the host: In this step, you need to free the memory that was allocated on the host. You can use the C free function to free the memory.

Steps for Matrix Multiplication using CUDA

Here are the steps for implementing matrix multiplication using CUDA C:

1. Matrix Initialization: The first step is to initialize the matrices that you want to multiply. You can use standard C or CUDA functions to allocate memory for the matrices and initialize their values. The matrices are usually represented as 2D arrays.

2. Memory Allocation: The next step is to allocate memory on the host and the device for the matrices. You can use the standard C malloc function to allocate memory on the host and the CUDA function cudaMalloc() to allocate memory on the device.
3. Data Transfer: The third step is to transfer data between the host and the device. You can use the CUDA function cudaMemcpy() to transfer data from the host to the device or vice versa.
4. Kernel Launch: The fourth step is to launch the CUDA kernel that will perform the matrix multiplication on the device. You can use the <<<...>>> syntax to specify the number of blocks and threads to use. Each thread in the kernel will compute one element of the output matrix.
5. Device Synchronization: The fifth step is to synchronize the device to ensure that all kernel executions have completed before proceeding. You can use the CUDA function cudaDeviceSynchronize() to synchronize the device.

6. Data Retrieval: The sixth step is to retrieve the result of the computation from the device to the host. You can use the CUDA function cudaMemcpy() to transfer data from the device to the host.
7. Memory Deallocation: The final step is to deallocate the memory that was allocated on the host and the device. You can use the C free function to deallocate memory on the host and the CUDA function cudaFree() to deallocate memory on the device.

QUESTIONS:

1. What are the advantages of using CUDA to perform vector addition and matrix multiplication compared to using a CPU?
2. How do you launch the CUDA kernel to perform different operations?.
3. How can you optimize the performance of the CUDA program for adding two large vectors and matrix multiplication.