**Awesome! Here’s Practical 4 as an individual full reply:**

**Practical 4: Vector Addition and Matrix Multiplication using CUDA C**

**Title:**

**Vector Addition and Matrix Multiplication using CUDA C**

**Introduction:**

As the need for high-performance computing grows, Graphics Processing Units (GPUs) have become essential due to their ability to perform massive parallel computations. CUDA (Compute Unified Device Architecture) is NVIDIA’s parallel computing platform and programming model, allowing developers to use GPUs for general-purpose processing. This practical focuses on two fundamental parallel computing problems—Vector Addition and Matrix Multiplication—using CUDA C. These problems demonstrate how data-parallelism can be leveraged to achieve substantial speedup over traditional CPU implementations.

**Objective:**

* To understand the basics of GPU programming with CUDA.
* To implement parallel vector addition using CUDA C.
* To implement parallel matrix multiplication using CUDA C.
* To compare the performance of CPU and GPU implementations.
* To develop hands-on skills in parallel programming with CUDA.

**Theory:**

**CUDA Overview:**

CUDA C extends C/C++ by allowing the programmer to define "kernels," functions that run on the GPU. CUDA supports thousands of threads running in parallel, making it highly efficient for data-parallel problems.

**Basic Concepts in CUDA:**

* **Host:** Refers to the CPU and its memory.
* **Device:** Refers to the GPU and its memory.
* **Kernel:** A function executed on the GPU.
* **Threads:** Smallest unit of execution.
* **Blocks and Grids:** Threads are organized into blocks, and blocks into grids.

**Vector Addition:**

Vector addition is one of the simplest parallel operations, where two vectors A and B are added element-wise to produce a resultant vector C.

**Mathematical Representation:** C[i]=A[i]+B[i]fori=0 to n−1C[i] = A[i] + B[i] \quad \text{for} \quad i = 0 \text{ to } n-1

**Sequential Vector Addition:**

for (int i = 0; i < n; i++) {

C[i] = A[i] + B[i];

}

**Parallel Vector Addition using CUDA:** Each thread handles one element of the vector addition:

\_\_global\_\_ void vectorAdd(int \*A, int \*B, int \*C, int n) {

int i = blockDim.x \* blockIdx.x + threadIdx.x;

if (i < n)

C[i] = A[i] + B[i];

}

**Steps:**

1. Allocate memory on the device.
2. Copy input data from host to device.
3. Launch the kernel with appropriate grid and block dimensions.
4. Copy result back from device to host.
5. Free device memory.

**Matrix Multiplication:**

Matrix multiplication involves computing the dot product of the rows of the first matrix with the columns of the second matrix.

**Mathematical Representation:** C[i][j]=∑k=0n−1A[i][k]×B[k][j]C[i][j] = \sum\_{k=0}^{n-1} A[i][k] \times B[k][j]

Where:

* A is of size (m × n)
* B is of size (n × p)
* Result C is of size (m × p)

**Sequential Matrix Multiplication:**

for (int i = 0; i < m; i++) {

for (int j = 0; j < p; j++) {

C[i][j] = 0;

for (int k = 0; k < n; k++) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

**Parallel Matrix Multiplication using CUDA:**

Each thread computes one element of the output matrix.

**Basic Kernel:**

\_\_global\_\_ void matrixMul(int \*A, int \*B, int \*C, int m, int n, int p) {

int row = blockIdx.y \* blockDim.y + threadIdx.y;

int col = blockIdx.x \* blockDim.x + threadIdx.x;

if (row < m && col < p) {

int value = 0;

for (int k = 0; k < n; ++k) {

value += A[row \* n + k] \* B[k \* p + col];

}

C[row \* p + col] = value;

}

}

**Optimization Techniques:**

* Using shared memory for faster data access.
* Tiling to improve memory coalescence and minimize global memory access.

**Applications of Vector Addition and Matrix Multiplication:**

* Physics simulations.
* Machine learning and deep learning (especially matrix operations).
* Computer graphics and image processing.
* Big data analytics.

**Advantages of CUDA Programming:**

* Massive parallelism leading to significant speedups.
* Better performance for data-parallel tasks.
* Scalability across different generations of NVIDIA GPUs.
* Access to sophisticated memory models (global, shared, texture memory).

**Conclusion:**

In this practical, we successfully implemented parallel vector addition and matrix multiplication using CUDA C. By utilizing the massive parallelism of GPUs, we achieved efficient computation for operations that are traditionally computation-heavy on CPUs. Understanding the principles of CUDA programming is crucial for solving complex, real-world problems where performance and scalability are critical.

✅  
All 4 Practicals are now prepared separately for you!  
Would you also like me to give you a sample **index page** and **acknowledgment page** if you are submitting these as a full lab manual? 📑✨  
(Just say YES if you want!)