# HPC CUDA C TUTORIAL 12 REPORT

CS22B2015 – HARSHITH B

## *1. Introduction*

This report analyzes the performance of serial and parallel implementations for calculating the matrix multiplication of two matrices of size 10000x10000. A serial code written in C and a parallel code written in CUDA (.cu) were developed to perform this computation, and their execution times are compared to evaluate efficiency and potential speedup.

## *2. Serial Code Snippet*

```c
void multiply_matrices_serial(double matrix1[N][N], double matrix2[N][N], double result[N][N], FILE *file) {
    clock_t start, end;
    start = clock();

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result[i][j] = 0;
            for (int k = 0; k < N; k++) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
            // Print only for every 1000th column
            if (j % 1000 == 0) {
                printf("result[%d][%d]: %lf\n", i, j, result[i][j]);
            }
        }
    }

    end = clock();
    double time_spent =(double) end - start / CLOCKS_PER_SEC;
    printf("Serial Time: %f seconds\n", time_spent);
    fprintf(file, "Serial Time: %f\n", time_spent);
}
```

### 3. Parallel Code Snippet

```
__global__ void multiply_matrices(double *d_matrix1, double *d_matrix2, double *d_result) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;

    if (i < N && j < N) {
        int idx = i * N + j;
        d_result[idx] = d_matrix1[idx] * d_matrix2[idx];
        printf("d_result[%d] = %f\n", idx, d_result[idx]);
    }
}
```

```
double *d_matrix1, *d_matrix2, *d_result;
cudaMalloc((void **)&d_matrix1, N * N * sizeof(double));
cudaMalloc((void **)&d_matrix2, N * N * sizeof(double));
cudaMalloc((void **)&d_result, N * N * sizeof(double));

cudaMemcpy(d_matrix1, matrix1, N * N * sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(d_matrix2, matrix2, N * N * sizeof(double), cudaMemcpyHostToDevice);

dim3 threadsPerBlock(16, 16);
dim3 blocksPerGrid((N + threadsPerBlock.x - 1) / threadsPerBlock.x,
                   (N + threadsPerBlock.y - 1) / threadsPerBlock.y);

cudaEvent_t start, end;
cudaEventCreate(&start);
cudaEventCreate(&end);

cudaEventRecord(start);
multiply_matrices<<<blocksPerGrid, threadsPerBlock>>>(d_matrix1, d_matrix2, d_result);
cudaEventRecord(end);
cudaEventSynchronize(end);

float milliseconds = 0;
cudaEventElapsedTime(&milliseconds, start, end);
printf("Time: %f seconds\n", milliseconds);

cudaMemcpy(result, d_result, N * N * sizeof(double), cudaMemcpyDeviceToHost);

cudaFree(d_matrix1);
cudaFree(d_matrix2);
cudaFree(d_result);
```

### 4. Terminal Output Screenshot

```
(venv) harshith@harshithb:~/Projects /SEM 6/HPC/tutorial-12$ nvcc -o parallel parallel.cu
(venv) harshith@harshithb:~/Projects /SEM 6/HPC/tutorial-12$ ./parallel
Time: 15.571456 seconds
```

```
result[9999][9000]: 2517257282330518.000000
Serial Time: 14939.116942 seconds
```

### *5. Serial and Parallel Code Execution Time and Speedup Calculation*

- **Serial Code Execution Time:** 14939.116942 seconds
- **Parallel Code Execution Time:** 15.571456 seconds

**Speedup Calculation:**

**Speedup = Serial Execution Time / Parallel Execution Time**

**Speedup Estimation = 959.39**