Assignment 01

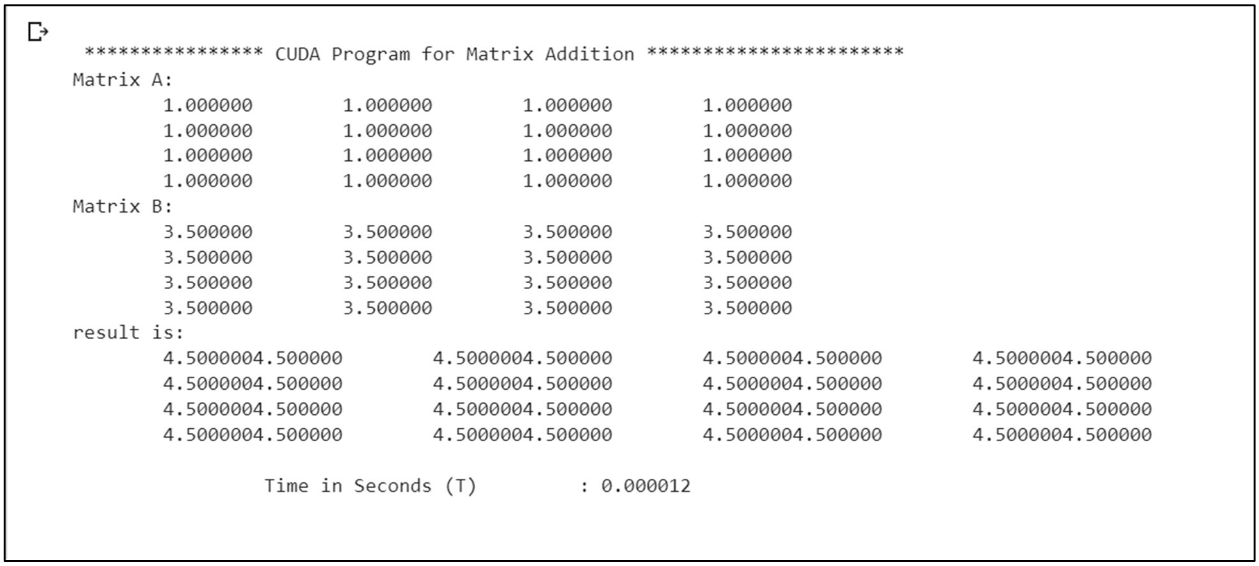
Program for Matrix Addition :

```
%%cu
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <string.h>
#include <cuda.h>
#include <assert.h>

const int N = 4;
const int blocksize = 2;

__global__ void add_matrix_on_gpu( float* a, float *b, float *c, int N )
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;
  int index = i + j*N;
  if ( i < N && j < N )
    c[index] = a[index] + b[index];
}

void add_matrix_on_cpu(float *a, float *b, float *d)
{
  int i;
  for(i = 0; i < N*N; i++)
  d[i] = a[i]+b[i];
}

int main()
{

  printf("\n **************** CUDA Program for Matrix Addition ***********
************ \n");
  float *a = new float[N*N];
  float *b = new float[N*N];
  float *c = new float[N*N];
  float *d = new float[N*N];

  for ( int i = 0; i < N*N; ++i ) {
    a[i] = 1.0f; b[i] = 3.5f; }
```

```
printf("Matrix A:\n");
for(int i=0; i<N*N; i++)
{
  printf("\t%f",a[i]);
  if((i+1)%N==0)
    printf("\n");
}

printf("Matrix B:\n");
for(int i=0; i<N*N; i++)
{
  printf("\t%f",b[i]);
  if((i+1)%N==0)
    printf("\n");
}
      struct timeval  TimeValue_Start;
      struct timezone TimeZone_Start;

      struct timeval  TimeValue_Final;
      struct timezone TimeZone_Final;
      long            time_start, time_end;
      double          time_overhead;


float *ad, *bd, *cd;
const int size = N*N*sizeof(float);

cudaMalloc( (void**)&ad, size );
cudaMalloc( (void**)&bd, size );
cudaMalloc( (void**)&cd, size );


cudaMemcpy( ad, a, size, cudaMemcpyHostToDevice );
cudaMemcpy( bd, b, size, cudaMemcpyHostToDevice );


dim3 dimBlock( blocksize, blocksize );
dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );

gettimeofday(&TimeValue_Start, &TimeZone_Start);
add_matrix_on_gpu<<<dimGrid, dimBlock>>>( ad, bd, cd, N );
      gettimeofday(&TimeValue_Final, &TimeZone_Final);


cudaMemcpy( c, cd, size, cudaMemcpyDeviceToHost );
```

```
  add_matrix_on_cpu(a,b,d);

        time_end = TimeValue_Final.tv_sec * 1000000 + TimeValue_Final.tv_u
sec;
        time_start = TimeValue_Start.tv_sec * 1000000 + TimeValue_Start.tv
_usec;

        time_overhead = (time_end - time_start)/1000000.0;

  printf("result is:\n");
  for(int i=0; i<N*N; i++)
  {
    printf("\t%f%f",c[i],d[i]);
    if((i+1)%N==0)
      printf("\n");
  }
  for(int i=0; i<N*N; i++)
  assert(c[i]==d[i]);

        printf("\n\t\t Time in Seconds (T)         : %lf\n\n",time_overhea
d);

  cudaFree( ad ); cudaFree( bd ); cudaFree( cd );
  delete[] a; delete[] b; delete[] c, delete[] d;
  return EXIT_SUCCESS;
}
```

Output :

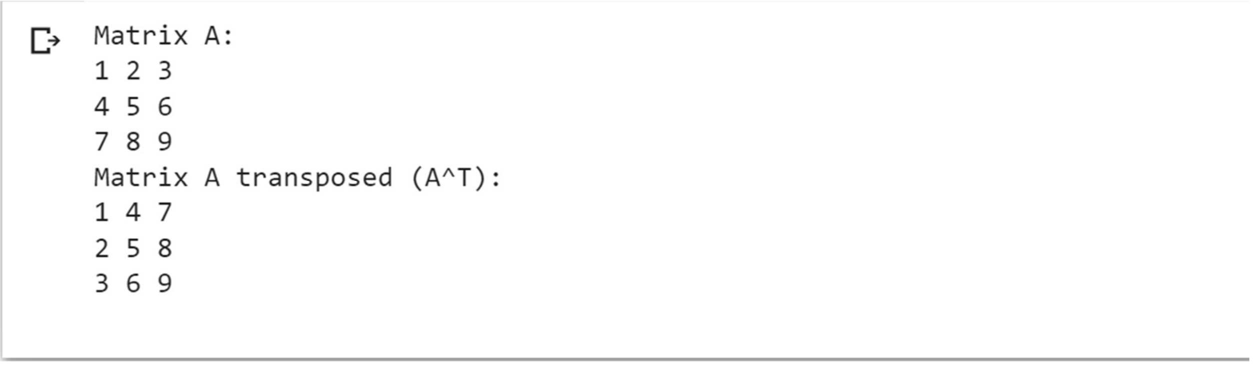


```
 **************** CUDA Program for Matrix Addition **********************
Matrix A:
        1.000000        1.000000        1.000000        1.000000
        1.000000        1.000000        1.000000        1.000000
        1.000000        1.000000        1.000000        1.000000
        1.000000        1.000000        1.000000        1.000000
Matrix B:
        3.500000        3.500000        3.500000        3.500000
        3.500000        3.500000        3.500000        3.500000
        3.500000        3.500000        3.500000        3.500000
        3.500000        3.500000        3.500000        3.500000
result is:
        4.5000004.500000        4.5000004.500000        4.5000004.500000        4.5000004.500000
        4.5000004.500000        4.5000004.500000        4.5000004.500000        4.5000004.500000
        4.5000004.500000        4.5000004.500000        4.5000004.500000        4.5000004.500000
        4.5000004.500000        4.5000004.500000        4.5000004.500000        4.5000004.500000

                 Time in Seconds (T)         : 0.000012
```

Program for transpose of matrix:

```
#include <cuda_runtime.h>
#include <iostream>

__global__ void transposeKernel(float* A, float* B, int m, int n) {
    int row = blockIdx.y*blockDim.y + threadIdx.y;
    int col = blockIdx.x*blockDim.x + threadIdx.x;
    if (row < n && col < m) {
        B[row*m + col] = A[col*n + row];
    }
}

int main() {
    int m = 3, n = 3;
    int size = m*n*sizeof(float);
    float* A, * B;
    cudaMalloc(&A, size);
    cudaMalloc(&B, size);
    // initialize A with sample values
    float A_host[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    cudaMemcpy(A, A_host, size, cudaMemcpyHostToDevice);
    dim3 block(3, 3);
    dim3 grid((m + block.x - 1) / block.x, (n + block.y - 1) / block.y);
    transposeKernel<<<grid, block>>>(A, B, m, n);
    cudaDeviceSynchronize();
    // print the input matrix A
    std::cout << "Matrix A:" << std::endl;
    for (int i = 0; i < 9; i++) {
        std::cout << A_host[i] << " ";
        if ((i + 1) % 3 == 0) std::cout << std::endl;
    }
    // print the transposed matrix B
    float B_host[9];
    cudaMemcpy(B_host, B, size, cudaMemcpyDeviceToHost);
    std::cout << "Matrix A transposed (A^T):" << std::endl;
    for (int i = 0; i < 9; i++) {
        std::cout << B_host[i] << " ";
        if ((i + 1) % 3 == 0) std::cout << std::endl;
    }
    cudaFree(A);
    cudaFree(B);
    return 0;
```

Output:



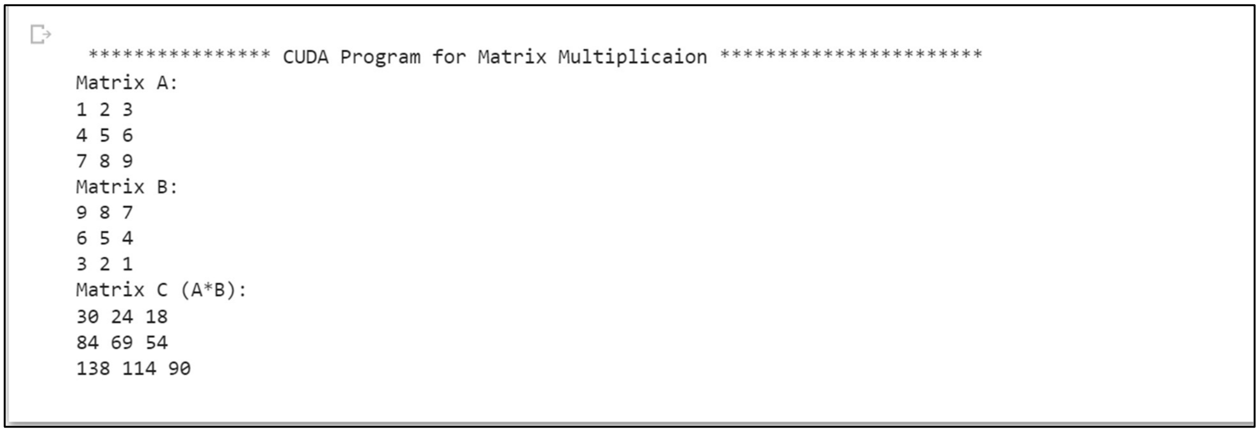
```
Matrix A:
1 2 3
4 5 6
7 8 9
Matrix A transposed (A^T):
1 4 7
2 5 8
3 6 9
```

Program for Matrix Multiplication :

```
%%cu
#include <cuda_runtime.h>
#include <iostream>

__global__ void matmulKernel(float* A, float* B, float* C) {
    int row = blockIdx.y*blockDim.y + threadIdx.y;
    int col = blockIdx.x*blockDim.x + threadIdx.x;
    if(row < 3 && col < 3) {
        float value = 0;
        for (int i = 0; i < 3; i++) {
            value += A[row*3 + i] * B[i*3 + col];
        }
        C[row*3 + col] = value;
    }
}

int main() {
    float* A, * B, * C;
    cudaMalloc(&A, 9 * sizeof(float));
    cudaMalloc(&B, 9 * sizeof(float));
    cudaMalloc(&C, 9 * sizeof(float));
    // initialize A and B with sample values
    float A_host[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    float B_host[9] = {9, 8, 7, 6, 5, 4, 3, 2, 1};
    cudaMemcpy(A, A_host, 9 * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(B, B_host, 9 * sizeof(float), cudaMemcpyHostToDevice);
    dim3 block(3, 3);
    dim3 grid((3 + block.x - 1) / block.x, (3 + block.y - 1) / block.y);
    matmulKernel<<<grid, block>>>(A, B, C);
    cudaDeviceSynchronize();
    std::cout<<"\n **************** CUDA Program for Matrix Multiplicaion
*********************** \n";
    // print the input matrices A and B
    std::cout << "Matrix A:" << std::endl;
    for (int i = 0; i < 9; i++) {
        std::cout << A_host[i] << " ";
        if ((i + 1) % 3 == 0) std::cout << std::endl;
    }
    std::cout << "Matrix B:" << std::endl;
    for (int i = 0; i < 9; i++) {
        std::cout << B_host[i] << " ";
        if ((i + 1) % 3 == 0) std::cout << std::endl;
    }
    // print the result matrix C
    float C_host[9];
    cudaMemcpy(C_host, C, 9 * sizeof(float), cudaMemcpyDeviceToHost);
    std::cout << "Matrix C (A*B):" << std::endl;
    for (int i = 0; i < 9; i++) {
        std::cout << C_host[i] << " ";
        if ((i + 1) % 3 == 0) std::cout << std::endl;
    }
    cudaFree(A);
    cudaFree(B);
    cudaFree(C);
    return 0;
}
```

Output:

-



```
 **************** CUDA Program for Matrix Multiplicaion ***********************
Matrix A:
1 2 3
4 5 6
7 8 9
Matrix B:
9 8 7
6 5 4
3 2 1
Matrix C (A*B):
30 24 18
84 69 54
138 114 90
```