# CS 461

## Lab Assignment 8

Name: Gandhi Dhruv Vipulkumar

Institute ID: 202151053

Date: 22-10-2024

**NOTE**: Due to unavailability of NVIDIA GPU in the local machine the following code is run on google colab.

Link: https://colab.research.google.com/drive/1ztJ8tsSUh5BT5raDCKg8Tmfa3-HYcn_m?usp=sharing

## Q. Implement Matrix Multiplication using CUDA

```
!apt-get install nvidia-cuda-toolkit g++

%%writefile matrix_mul.cu
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <cuda_runtime.h>
#include <omp.h>

#define BLOCK_SIZE 16
#define PRINT_LIMIT 10 // Limit to print elements of large matrices

// Function to print a matrix (with limits for large matrices)
void print_matrix(int* matrix, int rows, int cols, const char* name)
{
    printf("Matrix %s (%d x %d):\n", name, rows, cols);
    for (int i = 0; i < rows && i < PRINT_LIMIT; ++i)
    {
        for (int j = 0; j < cols && j < PRINT_LIMIT; ++j)
        {
            printf("%4d ", matrix[i * cols + j]);
        }
        if (cols > PRINT_LIMIT)
        {
            printf("... "); // Print ellipsis if there are more
columns
```

```
        }
        printf("\n");
    }
    if (rows > PRINT_LIMIT)
    {
        printf("... \n"); // Print ellipsis if there are more rows
    }
    printf("\n");
}

// CUDA kernel for general matrix multiplication
__global__ void gpu_matrix_mult(int* a, int* b, int* c, int m, int
n, int k)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int sum = 0;
    if (col < k && row < m)
    {
        for (int i = 0; i < n; i++)
        {
            sum += a[row * n + i] * b[i * k + col];
        }
        c[row * k + col] = sum;
    }
}

// CUDA kernel for square matrix multiplication
__global__ void gpu_square_matrix_mult(int* d_a, int* d_b, int*
d_result, int n)
{
    __shared__ int tile_a[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ int tile_b[BLOCK_SIZE][BLOCK_SIZE];

    int row = blockIdx.y * BLOCK_SIZE + threadIdx.y;
    int col = blockIdx.x * BLOCK_SIZE + threadIdx.x;
    int tmp = 0;
    int idx;

    for (int sub = 0; sub < gridDim.x; ++sub)
    {
        idx = row * n + sub * BLOCK_SIZE + threadIdx.x;
        tile_a[threadIdx.y][threadIdx.x] = (idx < n * n) ? d_a[idx]
: 0;

        idx = (sub * BLOCK_SIZE + threadIdx.y) * n + col;
        tile_b[threadIdx.y][threadIdx.x] = (idx < n * n) ? d_b[idx] : 0;

        __syncthreads();

        for (int k = 0; k < BLOCK_SIZE; ++k)
        {
            tmp += tile_a[threadIdx.y][k] * tile_b[k][threadIdx.x];
        }
        __syncthreads();
    }

    if (row < n && col < n)
    {
        d_result[row * n + col] = tmp;
    }
}

// OpenMP function for matrix multiplication (parallelized)
void openmp_matrix_mult(int* h_a, int* h_b, int* h_c, int m, int n, int k)
{
#pragma omp parallel for collapse(2)
    for (int i = 0; i < m; ++i)
    {
        for (int j = 0; j < k; ++j)
        {
            int tmp = 0;
            for (int h = 0; h < n; ++h)
            {
                tmp += h_a[i * n + h] * h_b[h * k + j];
            }
            h_c[i * k + j] = tmp;
        }
    }
}

// Normal (sequential) matrix multiplication function
void cpu_matrix_mult(int* h_a, int* h_b, int* h_result, int m, int n, int k)
{
    for (int i = 0; i < m; ++i)
    {
        for (int j = 0; j < k; ++j)
        {
            int tmp = 0;
            for (int h = 0; h < n; ++h)
            {
                tmp += h_a[i * n + h] * h_b[h * k + j];
            }
            h_result[i * k + j] = tmp;
        }
    }
}

// Main function
int main(int argc, char const* argv[])
{
    int m, n, k;
    srand(3333); // Fixed seed
    printf("Please type in m, n, and k: ");
    scanf("%d %d %d", &m, &n, &k);

    // Allocate memory in host RAM
    int* h_a, * h_b, * h_c, * h_cc;
    cudaMallocHost((void**)&h_a, sizeof(int) * m * n);
    cudaMallocHost((void**)&h_b, sizeof(int) * n * k);
    cudaMallocHost((void**)&h_c, sizeof(int) * m * k);
    cudaMallocHost((void**)&h_cc, sizeof(int) * m * k);

    // Random initialize matrix A
    for (int i = 0; i < m; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            h_a[i * n + j] = rand() % 1024;
        }
    }

    // Random initialize matrix B
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < k; ++j)
        {
            h_b[i * k + j] = rand() % 1024;
        }
    }

    // Print matrices A and B
    print_matrix(h_a, m, n, "A");
    print_matrix(h_b, n, k, "B");
    float gpu_elapsed_time_ms, cpu_elapsed_time_ms,
normal_elapsed_time_ms;

    // Start measuring GPU execution time
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start, 0);

    // Allocate memory space on the device
    int* d_a, * d_b, * d_c;
    cudaMalloc((void**)&d_a, sizeof(int) * m * n);
    cudaMalloc((void**)&d_b, sizeof(int) * n * k);
    cudaMalloc((void**)&d_c, sizeof(int) * m * k);

    // Copy matrix A and B from host to device memory
    cudaMemcpy(d_a, h_a, sizeof(int) * m * n,
cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, sizeof(int) * n * k,
cudaMemcpyHostToDevice);

    unsigned int grid_rows = (m + BLOCK_SIZE - 1) / BLOCK_SIZE;
    unsigned int grid_cols = (k + BLOCK_SIZE - 1) / BLOCK_SIZE;
    dim3 dimGrid(grid_cols, grid_rows);
    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);

    // Launch the appropriate kernel
    if (m == n && n == k)
    {
        gpu_square_matrix_mult << <dimGrid, dimBlock >> > (d_a, d_b,
d_c, n);
    }
    else
    {
        gpu_matrix_mult << <dimGrid, dimBlock >> > (d_a, d_b, d_c,
m, n, k);
    }

    // Transfer results from device to host
    cudaMemcpy(h_c, d_c, sizeof(int) * m * k,
cudaMemcpyDeviceToHost);
    cudaDeviceSynchronize(); // Wait for GPU to finish
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);

    // Compute time elapsed on GPU computing
    cudaEventElapsedTime(&gpu_elapsed_time_ms, start, stop);
    printf("Time elapsed on matrix multiplication of %dx%d . %dx%d
on GPU: %f ms.\n", m, n, n, k, gpu_elapsed_time_ms);

    // Print result matrix C (GPU result)
    print_matrix(h_c, m, k, "C (GPU Result)");

    // Start measuring normal (sequential) execution time
    double start_time = omp_get_wtime();
    cpu_matrix_mult(h_a, h_b, h_cc, m, n, k);
    double end_time = omp_get_wtime();
    normal_elapsed_time_ms = (end_time - start_time) * 1000.0; //
Convert to milliseconds
    printf("Time elapsed on normal matrix multiplication of %dx%d .
%dx%d on CPU: %f ms.\n", m, n, n, k, normal_elapsed_time_ms);

    // Print result matrix C (CPU result)
    print_matrix(h_cc, m, k, "C (CPU Result)");

    // Start measuring CPU execution time using OpenMP
    start_time = omp_get_wtime();
    openmp_matrix_mult(h_a, h_b, h_cc, m, n, k);
    end_time = omp_get_wtime();
    cpu_elapsed_time_ms = (end_time - start_time) * 1000.0; //
Convert to milliseconds
    printf("Time elapsed on matrix multiplication of %dx%d . %dx%d
on CPU (OpenMP): %f ms.\n", m, n, n, k, cpu_elapsed_time_ms);

    // Compare the results
    int all_ok = 1;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < k; j++)
        {
            if (h_cc[i * k + j] != h_c[i * k + j])
            {
                all_ok = 0;
                printf("Mismatch at [%d][%d]: GPU=%d, CPU=%d\n", i,
j, h_c[i * k + j], h_cc[i * k + j]);
                break;
            }
        }
        if (!all_ok) break;
    }

    printf("Matrix multiplication %s\n", all_ok ? "successful!" :
```

```
"failed.");

    // Free GPU memory
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    // Free CPU memory
    cudaFreeHost(h_a);
    cudaFreeHost(h_b);
    cudaFreeHost(h_c);
    cudaFreeHost(h_cc);

    return 0;
}
```

## Code Explanation:

**1. Includes and Defines**:

- Includes standard libraries for input/output, memory management, and CUDA runtime.
- Defines BLOCK_SIZE, which sets the dimensions of the blocks used in the CUDA kernel.

**2. CUDA Kernels**:

- **gpu_matrix_mult**: A kernel for general matrix multiplication. It computes the value for each element of the result matrix C based on matrices A and B.
- **gpu_square_matrix_mult**: An optimized kernel for square matrices that uses shared memory for better performance. It loads sub-matrices (tiles) into shared memory, reducing global memory accesses.

**3. Matrix Multiplication Functions**:

- **openmp_matrix_mult**: Uses OpenMP for parallel matrix multiplication on the CPU. It employs nested loops with #pragma omp parallel for collapse(2) to parallelize both outer loops.
- **cpu_matrix_mult**: A normal sequential implementation for matrix multiplication on the CPU.

4. **Main Function**:

- Reads the dimensions of the matrices from the user.
- Allocates memory for matrices A, B, and results C on both host (CPU) and device (GPU).
- Initializes matrices A and B with random values.
- Measures and prints the time taken for matrix multiplication on the GPU and CPU.
- Compares the results from GPU and CPU computations for correctness.
- Frees the allocated memory.

## Key Features:

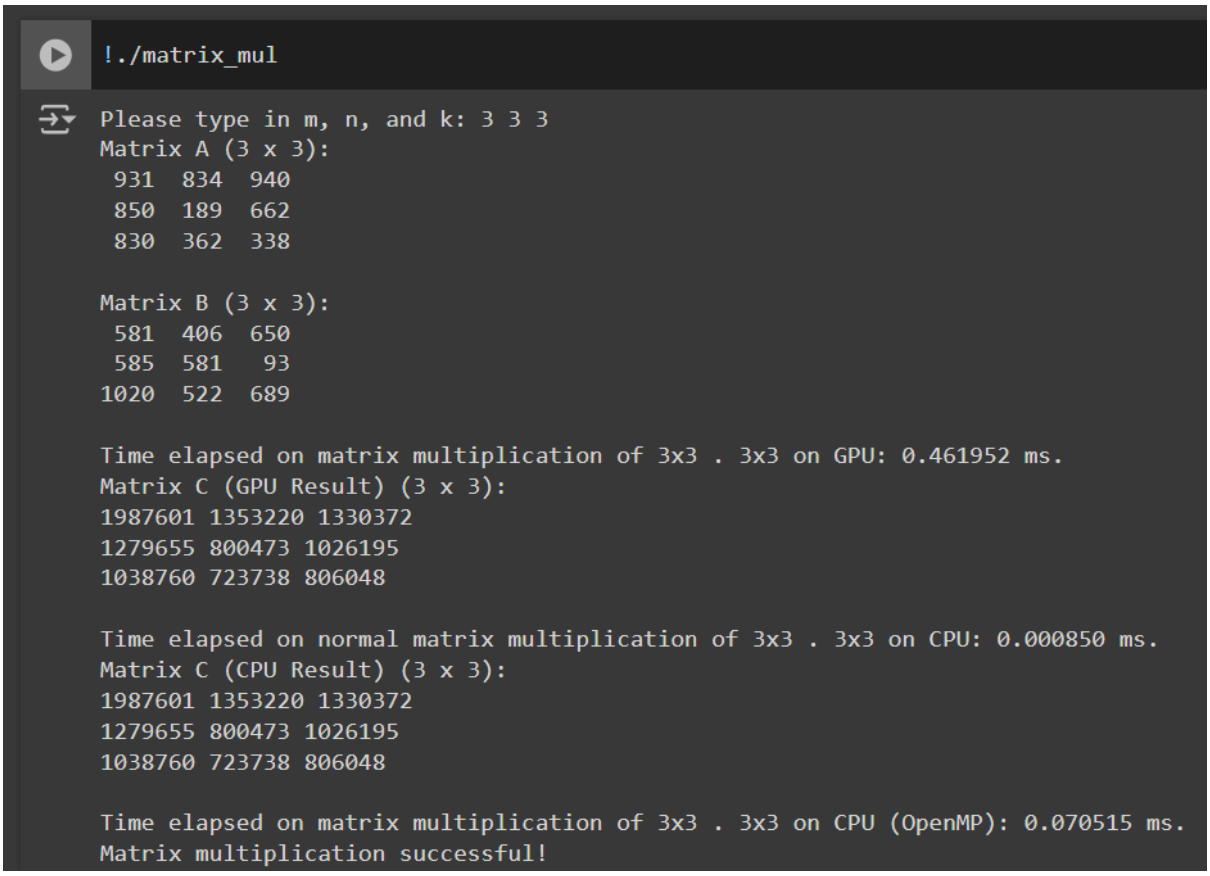
1. **CUDA Implementation**:
   - The code utilizes CUDA for GPU acceleration, enabling efficient handling of matrix multiplication tasks, especially for larger matrices.
2. **Optimized Memory Usage**:
   - Uses shared memory in the gpu_square_matrix_mult kernel to speed up memory access times by reducing global memory accesses.
3. **Parallelization**:
   - The code showcases different levels of parallelization: GPU-based with CUDA and CPU-based using OpenMP.
4. **Performance Measurement**:
   - It measures the execution time for GPU and CPU matrix multiplication, allowing for performance comparisons.
5. **Validation**:
   - The code checks the results of the matrix multiplication between GPU and CPU to ensure correctness, printing mismatches if found.
6. **Dynamic Input**:
   - The dimensions of the matrices are provided by the user at runtime, making the program flexible for different sizes of matrices.

7. **User-Friendly Output**:

- It prints the matrices before multiplication and shows the resulting matrices for both GPU and CPU, making it easy to verify results.

## Testing Phase:

### 1) multiplication for very small matrices(3x3):

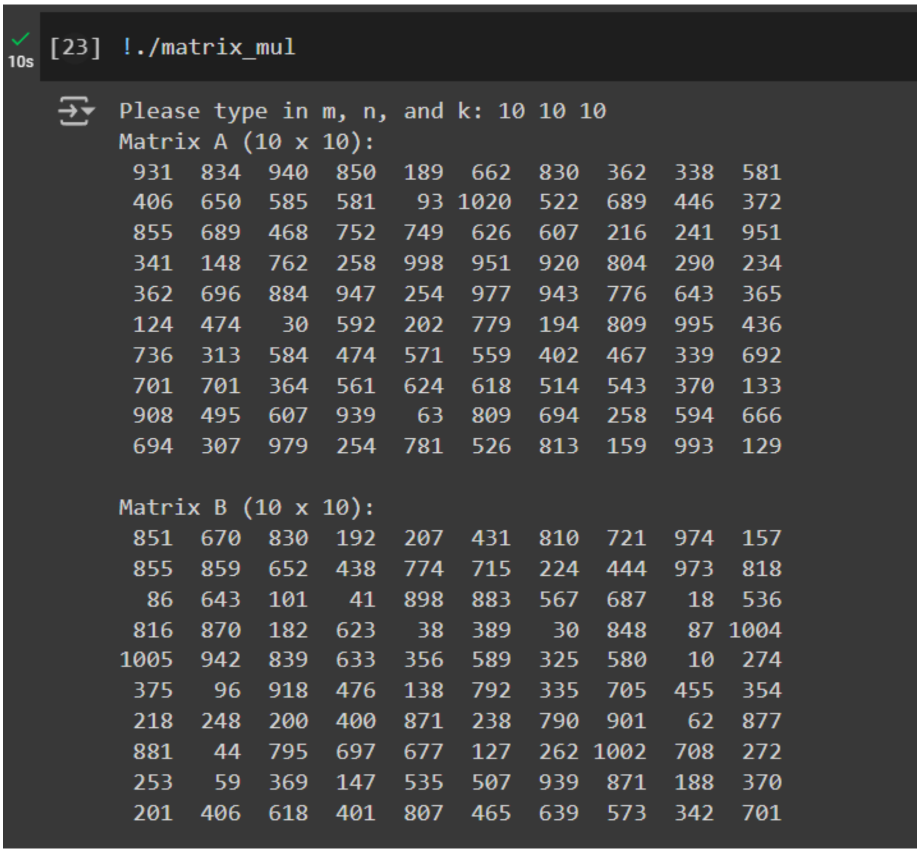


```
!./matrix_mul
```

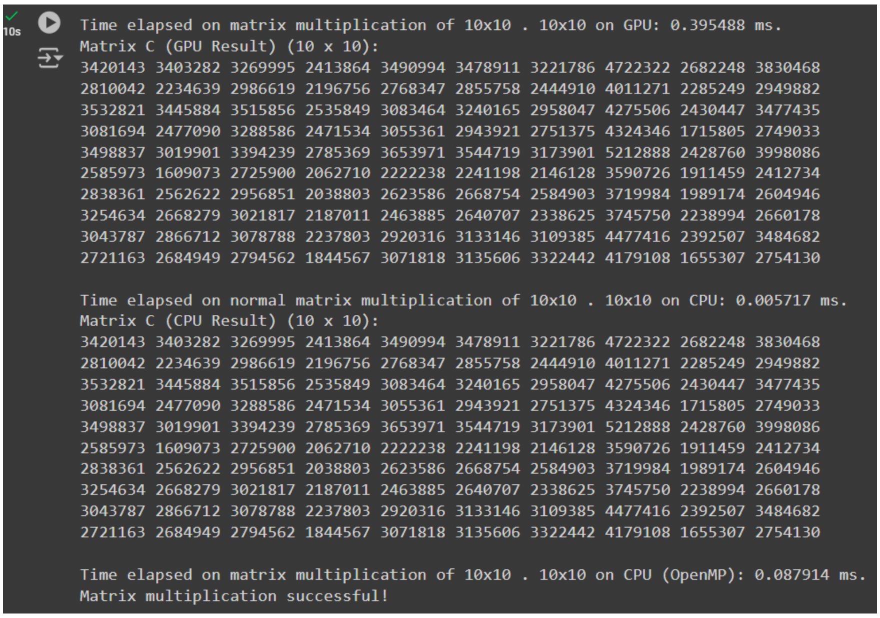
```
Please type in m, n, and k: 3 3 3
Matrix A (3 x 3):
 931  834  940
 850  189  662
 830  362  338

Matrix B (3 x 3):
 581  406  650
 585  581   93
1020  522  689

Time elapsed on matrix multiplication of 3x3 . 3x3 on GPU: 0.461952 ms.
Matrix C (GPU Result) (3 x 3):
1987601 1353220 1330372
1279655 800473 1026195
1038760 723738 806048

Time elapsed on normal matrix multiplication of 3x3 . 3x3 on CPU: 0.000850 ms.
Matrix C (CPU Result) (3 x 3):
1987601 1353220 1330372
1279655 800473 1026195
1038760 723738 806048

Time elapsed on matrix multiplication of 3x3 . 3x3 on CPU (OpenMP): 0.070515 ms.
Matrix multiplication successful!
```

- Time elapsed on matrix multiplication of 3x3 . 3x3 on GPU**: 0.461952 ms.**
- Time elapsed on normal matrix multiplication of 3x3 . 3x3 on CPU: **0.000850 ms.**
- Time elapsed on matrix multiplication of 3x3 . 3x3 on CPU (OpenMP): **0.070515 ms.**

### 2) Multiplication for small matrices(10x10):



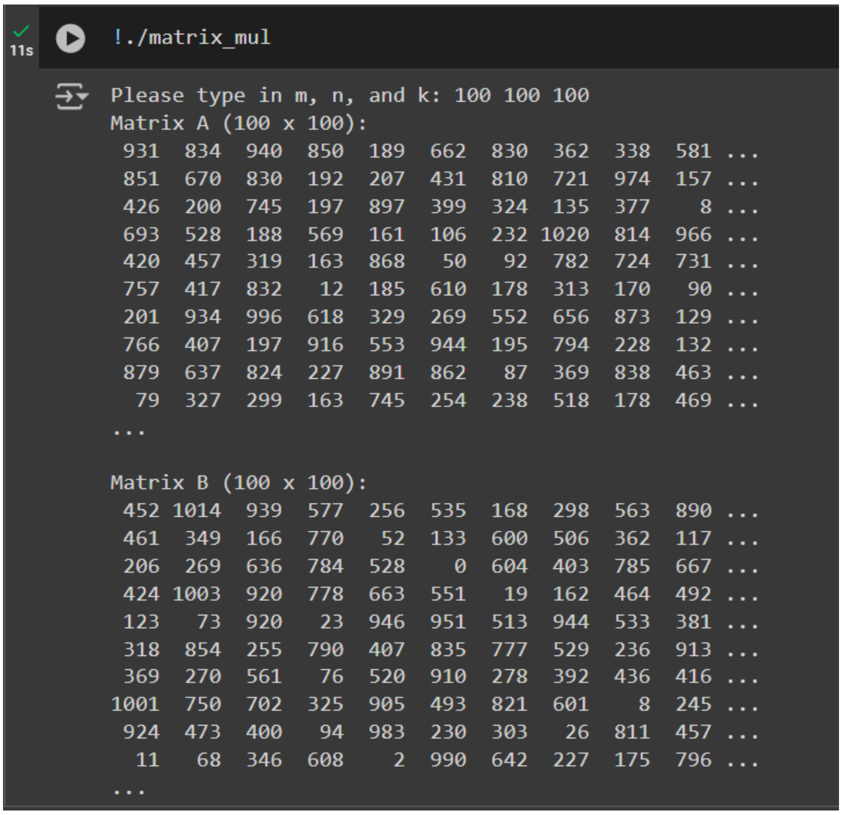
```
[23] !./matrix_mul

Please type in m, n, and k: 10 10 10
Matrix A (10 x 10):
 931  834  940  850  189  662  830  362  338  581
 406  650  585  581   93 1020  522  689  446  372
 855  689  468  752  749  626  607  216  241  951
 341  148  762  258  998  951  920  804  290  234
 362  696  884  947  254  977  943  776  643  365
 124  474   30  592  202  779  194  809  995  436
 736  313  584  474  571  559  402  467  339  692
 701  701  364  561  624  618  514  543  370  133
 908  495  607  939   63  809  694  258  594  666
 694  307  979  254  781  526  813  159  993  129

Matrix B (10 x 10):
 851  670  830  192  207  431  810  721  974  157
 855  859  652  438  774  715  224  444  973  818
  86  643  101   41  898  883  567  687   18  536
 816  870  182  623   38  389   30  848   87 1004
1005  942  839  633  356  589  325  580   10  274
 375   96  918  476  138  792  335  705  455  354
 218  248  200  400  871  238  790  901   62  877
 881   44  795  697  677  127  262 1002  708  272
 253   59  369  147  535  507  939  871  188  370
 201  406  618  401  807  465  639  573  342  701
```



```
Time elapsed on matrix multiplication of 10x10 . 10x10 on GPU: 0.395488 ms.
Matrix C (GPU Result) (10 x 10):
3420143 3403282 3269995 2413864 3490994 3478911 3221786 4722322 2682248 3830468
2810042 2234639 2986619 2196756 2768347 2855758 2444910 4011271 2285249 2949882
3532821 3445884 3515856 2535849 3083464 3240165 2958047 4275506 2430447 3477435
3081694 2477090 3288586 2471534 3055361 2943921 2751375 4324346 1715805 2749033
3498837 3019901 3394239 2785369 3653971 3544719 3173901 5212888 2428760 3998086
2585973 1609073 2725900 2062710 2222238 2241198 2146128 3590726 1911459 2412734
2838361 2562622 2956851 2038803 2623586 2668754 2584903 3719984 1989174 2604946
3254634 2668279 3021817 2187011 2463885 2640707 2338625 3745750 2238994 2660178
3043787 2866712 3078788 2237803 2920316 3133146 3109385 4477416 2392507 3484682
2721163 2684949 2794562 1844567 3071818 3135606 3322442 4179108 1655307 2754130

Time elapsed on normal matrix multiplication of 10x10 . 10x10 on CPU: 0.005717 ms.
Matrix C (CPU Result) (10 x 10):
3420143 3403282 3269995 2413864 3490994 3478911 3221786 4722322 2682248 3830468
2810042 2234639 2986619 2196756 2768347 2855758 2444910 4011271 2285249 2949882
3532821 3445884 3515856 2535849 3083464 3240165 2958047 4275506 2430447 3477435
3081694 2477090 3288586 2471534 3055361 2943921 2751375 4324346 1715805 2749033
3498837 3019901 3394239 2785369 3653971 3544719 3173901 5212888 2428760 3998086
2585973 1609073 2725900 2062710 2222238 2241198 2146128 3590726 1911459 2412734
2838361 2562622 2956851 2038803 2623586 2668754 2584903 3719984 1989174 2604946
3254634 2668279 3021817 2187011 2463885 2640707 2338625 3745750 2238994 2660178
3043787 2866712 3078788 2237803 2920316 3133146 3109385 4477416 2392507 3484682
2721163 2684949 2794562 1844567 3071818 3135606 3322442 4179108 1655307 2754130

Time elapsed on matrix multiplication of 10x10 . 10x10 on CPU (OpenMP): 0.087914 ms.
Matrix multiplication successful!
```

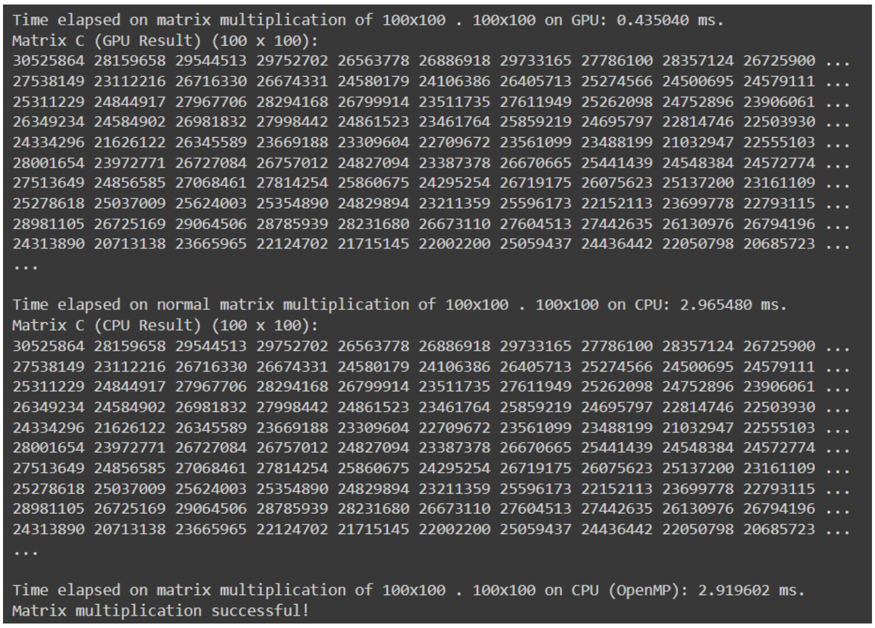
- Time elapsed on matrix multiplication of 10x10 . 10x10 on GPU: **0.395488 ms.**
- Time elapsed on normal matrix multiplication of 10x10 . 10x10 on CPU: **0.005717 ms.**
- Time elapsed on matrix multiplication of 10x10 . 10x10 on CPU (OpenMP): **0.087914 ms.**

### 3) Increase the size of matrices(100x100):



```
!./matrix_mul
```

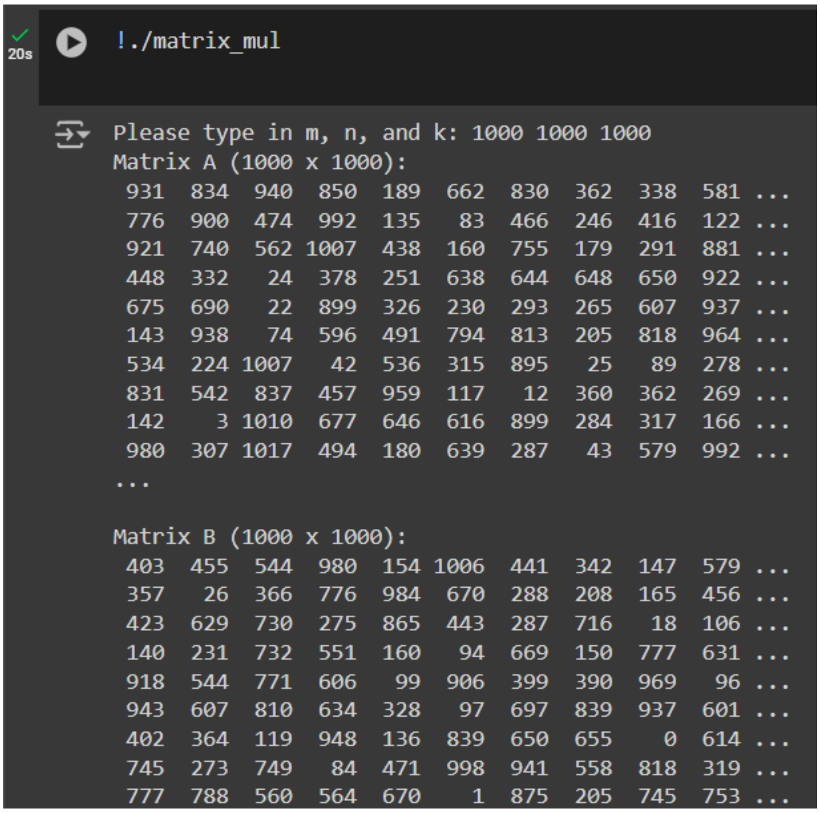
```
Please type in m, n, and k: 100 100 100
Matrix A (100 x 100):
 931  834  940  850  189  662  830  362  338  581 ...
 851  670  830  192  207  431  810  721  974  157 ...
 426  200  745  197  897  399  324  135  377    8 ...
 693  528  188  569  161  106  232 1020  814  966 ...
 420  457  319  163  868   50   92  782  724  731 ...
 757  417  832   12  185  610  178  313  170   90 ...
 201  934  996  618  329  269  552  656  873  129 ...
 766  407  197  916  553  944  195  794  228  132 ...
 879  637  824  227  891  862   87  369  838  463 ...
  79  327  299  163  745  254  238  518  178  469 ...
...

Matrix B (100 x 100):
 452 1014  939  577  256  535  168  298  563  890 ...
 461  349  166  770   52  133  600  506  362  117 ...
 206  269  636  784  528    0  604  403  785  667 ...
 424 1003  920  778  663  551   19  162  464  492 ...
 123   73  920   23  946  951  513  944  533  381 ...
 318  854  255  790  407  835  777  529  236  913 ...
 369  270  561   76  520  910  278  392  436  416 ...
1001  750  702  325  905  493  821  601    8  245 ...
 924  473  400   94  983  230  303   26  811  457 ...
  11   68  346  608    2  990  642  227  175  796 ...
...
```



```
Time elapsed on matrix multiplication of 100x100 . 100x100 on GPU: 0.435040 ms.
Matrix C (GPU Result) (100 x 100):
30525864 28159658 29544513 29752702 26563778 26886918 29733165 27786100 28357124 26725900 ...
27538149 23112216 26716330 26674331 24580179 24106386 26405713 25274566 24500695 24579111 ...
25311229 24844917 27967706 28294168 26799914 23511735 27611949 25262098 24752896 23906061 ...
26349234 24584902 26981832 27998442 24861523 23461764 25859219 24695797 22814746 22503930 ...
24334296 21626122 26345589 23669188 23309604 22709672 23561099 23488199 21032947 22555103 ...
28001654 23972771 26727084 26757012 24827094 23387378 26670665 25441439 24548384 24572774 ...
27513649 24856585 27068461 27814254 25860675 24295254 26719175 26075623 25137200 23161109 ...
25278618 25037009 25624003 25354890 24829894 23211359 25596173 22152113 23699778 22793115 ...
28981105 26725169 29064506 28785939 28231680 26673110 27604513 27442635 26130976 26794196 ...
24313890 20713138 23665965 22124702 21715145 22002200 25059437 24436442 22050798 20685723 ...
...

Time elapsed on normal matrix multiplication of 100x100 . 100x100 on CPU: 2.965480 ms.
Matrix C (CPU Result) (100 x 100):
30525864 28159658 29544513 29752702 26563778 26886918 29733165 27786100 28357124 26725900 ...
27538149 23112216 26716330 26674331 24580179 24106386 26405713 25274566 24500695 24579111 ...
25311229 24844917 27967706 28294168 26799914 23511735 27611949 25262098 24752896 23906061 ...
26349234 24584902 26981832 27998442 24861523 23461764 25859219 24695797 22814746 22503930 ...
24334296 21626122 26345589 23669188 23309604 22709672 23561099 23488199 21032947 22555103 ...
28001654 23972771 26727084 26757012 24827094 23387378 26670665 25441439 24548384 24572774 ...
27513649 24856585 27068461 27814254 25860675 24295254 26719175 26075623 25137200 23161109 ...
25278618 25037009 25624003 25354890 24829894 23211359 25596173 22152113 23699778 22793115 ...
28981105 26725169 29064506 28785939 28231680 26673110 27604513 27442635 26130976 26794196 ...
24313890 20713138 23665965 22124702 21715145 22002200 25059437 24436442 22050798 20685723 ...
...

Time elapsed on matrix multiplication of 100x100 . 100x100 on CPU (OpenMP): 2.919602 ms.
Matrix multiplication successful!
```

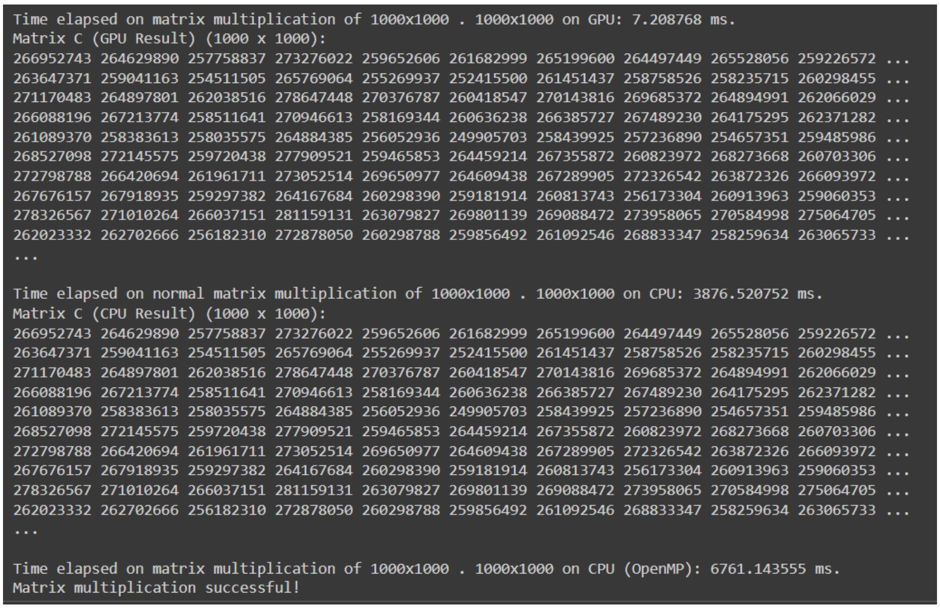
- Time elapsed on matrix multiplication of 100x100 . 100x100 on GPU: **0.435040 ms.**
- Time elapsed on normal matrix multiplication of 100x100 . 100x100 on CPU: **2.965480 ms.**
- Time elapsed on matrix multiplication of 100x100 . 100x100 on CPU (OpenMP): **2.919602 ms.**

## 4) Matrix multiplication for large matrices(1000x1000):



```
!./matrix_mul
```

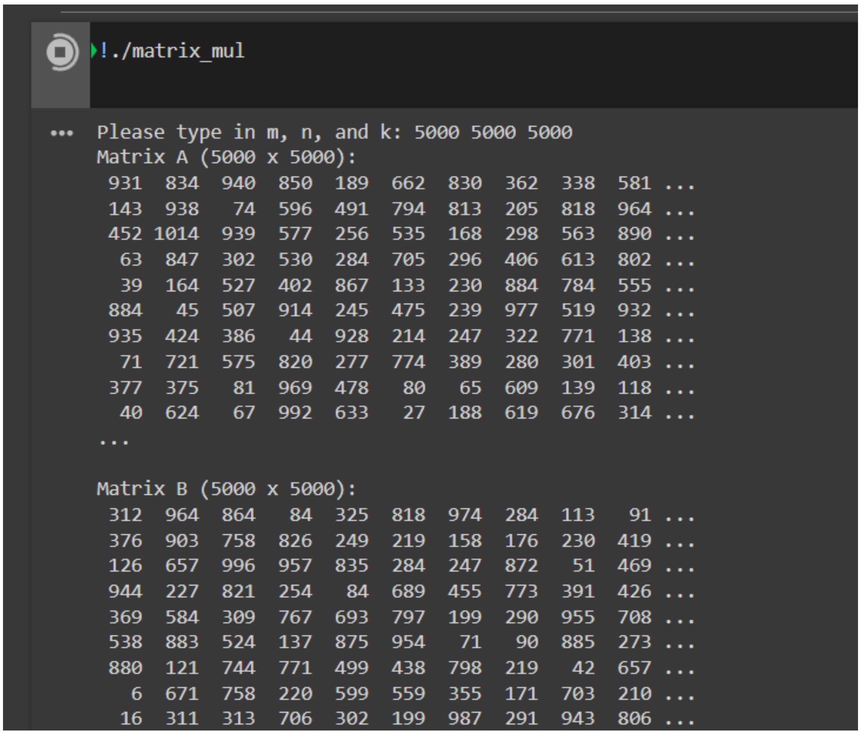
```
Please type in m, n, and k: 1000 1000 1000
Matrix A (1000 x 1000):
  931  834  940  850  189  662  830  362  338  581 ...
  776  900  474  992  135   83  466  246  416  122 ...
  921  740  562 1007  438  160  755  179  291  881 ...
  448  332   24  378  251  638  644  648  650  922 ...
  675  690   22  899  326  230  293  265  607  937 ...
  143  938   74  596  491  794  813  205  818  964 ...
  534  224 1007   42  536  315  895   25   89  278 ...
  831  542  837  457  959  117   12  360  362  269 ...
  142    3 1010  677  646  616  899  284  317  166 ...
  980  307 1017  494  180  639  287   43  579  992 ...
...

Matrix B (1000 x 1000):
  403  455  544  980  154 1006  441  342  147  579 ...
  357   26  366  776  984  670  288  208  165  456 ...
  423  629  730  275  865  443  287  716   18  106 ...
  140  231  732  551  160   94  669  150  777  631 ...
  918  544  771  606   99  906  399  390  969   96 ...
  943  607  810  634  328   97  697  839  937  601 ...
  402  364  119  948  136  839  650  655    0  614 ...
  745  273  749   84  471  998  941  558  818  319 ...
  777  788  560  564  670    1  875  205  745  753 ...
```



```
Time elapsed on matrix multiplication of 1000x1000 . 1000x1000 on GPU: 7.208768 ms.
Matrix C (GPU Result) (1000 x 1000):
266952743 264629890 257758837 273276022 259652606 261682999 265199600 264497449 265528056 259226572 ...
263647371 259041163 254511505 265769064 255269937 252415500 261451437 258758526 258235715 260298455 ...
271170483 264897801 262038516 278647448 270376787 260418547 270143816 269685372 264894991 262066029 ...
266088196 267213774 258511641 270946613 258169344 260636238 266385727 267489230 264175295 262371282 ...
261089370 258383613 258035575 264884385 256052936 249905703 258439925 257236890 254657351 259485986 ...
268527098 272145575 259720438 277909521 259465853 264459214 267355872 260823972 268273668 260703306 ...
272798788 266420694 261961711 273052514 269650977 264609438 267289905 272326542 263872326 266093972 ...
267676157 267918935 259297382 264167684 260298390 259181914 260813743 256173304 260913963 259060353 ...
278326567 271010264 266037151 281159131 263079827 269801139 269088472 273958065 270584998 275064705 ...
262023332 262702666 256182310 272878050 260298788 259856492 261092546 268833347 258259634 263065733 ...
...

Time elapsed on normal matrix multiplication of 1000x1000 . 1000x1000 on CPU: 3876.520752 ms.
Matrix C (CPU Result) (1000 x 1000):
266952743 264629890 257758837 273276022 259652606 261682999 265199600 264497449 265528056 259226572 ...
263647371 259041163 254511505 265769064 255269937 252415500 261451437 258758526 258235715 260298455 ...
271170483 264897801 262038516 278647448 270376787 260418547 270143816 269685372 264894991 262066029 ...
266088196 267213774 258511641 270946613 258169344 260636238 266385727 267489230 264175295 262371282 ...
261089370 258383613 258035575 264884385 256052936 249905703 258439925 257236890 254657351 259485986 ...
268527098 272145575 259720438 277909521 259465853 264459214 267355872 260823972 268273668 260703306 ...
272798788 266420694 261961711 273052514 269650977 264609438 267289905 272326542 263872326 266093972 ...
267676157 267918935 259297382 264167684 260298390 259181914 260813743 256173304 260913963 259060353 ...
278326567 271010264 266037151 281159131 263079827 269801139 269088472 273958065 270584998 275064705 ...
262023332 262702666 256182310 272878050 260298788 259856492 261092546 268833347 258259634 263065733 ...
...

Time elapsed on matrix multiplication of 1000x1000 . 1000x1000 on CPU (OpenMP): 6761.143555 ms.
Matrix multiplication successful!
```

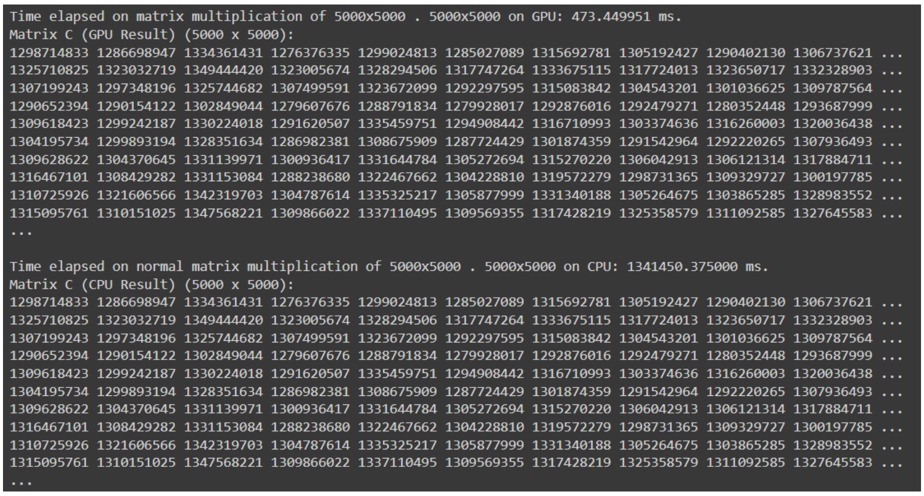
- Time elapsed on matrix multiplication of 1000x1000 . 1000x1000 on GPU: **7.208768 ms.**
- Time elapsed on normal matrix multiplication of 1000x1000 . 1000x1000 on CPU: **3876.520752 ms.**
- Time elapsed on matrix multiplication of 1000x1000 . 1000x1000 on CPU (OpenMP): **6761.143555 ms.**

## 5) Matrix Multiplication for very large matrices(5000x5000):



```
!./matrix_mul
```

```
Please type in m, n, and k: 5000 5000 5000
Matrix A (5000 x 5000):
 931  834  940  850  189  662  830  362  338  581 ...
 143  938   74  596  491  794  813  205  818  964 ...
 452 1014  939  577  256  535  168  298  563  890 ...
  63  847  302  530  284  705  296  406  613  802 ...
  39  164  527  402  867  133  230  884  784  555 ...
 884   45  507  914  245  475  239  977  519  932 ...
 935  424  386   44  928  214  247  322  771  138 ...
  71  721  575  820  277  774  389  280  301  403 ...
 377  375   81  969  478   80   65  609  139  118 ...
  40  624   67  992  633   27  188  619  676  314 ...
...

Matrix B (5000 x 5000):
 312  964  864   84  325  818  974  284  113   91 ...
 376  903  758  826  249  219  158  176  230  419 ...
 126  657  996  957  835  284  247  872   51  469 ...
 944  227  821  254   84  689  455  773  391  426 ...
 369  584  309  767  693  797  199  290  955  708 ...
 538  883  524  137  875  954   71   90  885  273 ...
 880  121  744  771  499  438  798  219   42  657 ...
   6  671  758  220  599  559  355  171  703  210 ...
  16  311  313  706  302  199  987  291  943  806 ...
```



```
Time elapsed on matrix multiplication of 5000x5000 . 5000x5000 on GPU: 473.449951 ms.
Matrix C (GPU Result) (5000 x 5000):
1298714833 1286698947 1334361431 1276376335 1299024813 1285027089 1315692781 1305192427 1290402130 1306737621 ...
1325710825 1323032719 1349444420 1323005674 1328294506 1317747264 1333675115 1317724013 1323650717 1332328903 ...
1307199243 1297348196 1325744682 1307499591 1323672099 1292297595 1315083842 1304543201 1301036625 1309787564 ...
1290652394 1290154122 1302849044 1279607676 1288791834 1279928017 1292876016 1292479271 1280352448 1293687999 ...
1309618423 1299242187 1330224018 1291620507 1335459751 1294908442 1316710993 1303374636 1316260003 1320036438 ...
1304195734 1299893194 1328351634 1286982381 1308675909 1287724429 1301874359 1291542964 1292220265 1307936493 ...
1309628622 1304370645 1331139971 1300936417 1331644784 1305272694 1315270220 1306042913 1306121314 1317884711 ...
1316467101 1308429282 1331153084 1288238680 1322467662 1304228810 1319572279 1298731365 1309329727 1300197785 ...
1310725926 1321606566 1342319703 1304787614 1335325217 1305877999 1331340188 1305264675 1303865285 1328983552 ...
1315095761 1310151025 1347568221 1309866022 1337110495 1309569355 1317428219 1325358579 1311092585 1327645583 ...
...

Time elapsed on normal matrix multiplication of 5000x5000 . 5000x5000 on CPU: 1341450.375000 ms.
Matrix C (CPU Result) (5000 x 5000):
1298714833 1286698947 1334361431 1276376335 1299024813 1285027089 1315692781 1305192427 1290402130 1306737621 ...
1325710825 1323032719 1349444420 1323005674 1328294506 1317747264 1333675115 1317724013 1323650717 1332328903 ...
1307199243 1297348196 1325744682 1307499591 1323672099 1292297595 1315083842 1304543201 1301036625 1309787564 ...
1290652394 1290154122 1302849044 1279607676 1288791834 1279928017 1292876016 1292479271 1280352448 1293687999 ...
1309618423 1299242187 1330224018 1291620507 1335459751 1294908442 1316710993 1303374636 1316260003 1320036438 ...
1304195734 1299893194 1328351634 1286982381 1308675909 1287724429 1301874359 1291542964 1292220265 1307936493 ...
1309628622 1304370645 1331139971 1300936417 1331644784 1305272694 1315270220 1306042913 1306121314 1317884711 ...
1316467101 1308429282 1331153084 1288238680 1322467662 1304228810 1319572279 1298731365 1309329727 1300197785 ...
1310725926 1321606566 1342319703 1304787614 1335325217 1305877999 1331340188 1305264675 1303865285 1328983552 ...
1315095761 1310151025 1347568221 1309866022 1337110495 1309569355 1317428219 1325358579 1311092585 1327645583 ...
...
```

- Time elapsed on matrix multiplication of 5000x5000 . 5000x5000 on GPU: **473.449951 ms.**
- Time elapsed on normal matrix multiplication of 5000x5000 . 5000x5000 on CPU: **1341450.375000 ms.**

**Conclusion:** Successfully computed Matrix Multiplication using CUDA and we can clearly observe that for large matrices, the matrix multiplication on GPU is much faster than CPU.