Assignment – 4

1. Sum of two large vectors using CUDA.

Code:

```
#include <iostream>
#include <vector>
#include <chrono>
#include <cstdlib>

using namespace std;

// CUDA kernel to perform vector addition
__global__ void vectorAdd(int *a, int *b, int *c, int n) {
   int tid = blockIdx.x * blockDim.x + threadIdx.x;
   if (tid < n)
      c[tid] = a[tid] + b[tid];
}


// Function to copy result back to host
void copyResult(vector<int> &c, int *d_c, int n) {
   cudaMemcpy(c.data(), d_c, n * sizeof(int), cudaMemcpyDeviceToHost);
}

void sum(const vector<int> &a, const vector<int> &b, vector<int> &d, int n)
{
   for(int i =0; i<n; i++)
   {
      d[i] = a[i] + b[i];
   }
}

double timeCal() {
   using namespace chrono;
   return duration_cast<duration<double>>(steady_clock::now().time_since_epoch()).count();
}


int main() {
   int n;
   cout << "Enter the size of the vectors: ";
   cin >> n;

   // Host vectors
   vector<int> a(n), b(n), c(n), d(n);

   // alloting random values from range 0 to 1000
   for (int i = 0; i < n; i++) {
      a[i] = rand() % 1000;
      b[i] = rand() % 1000;
   }
```

```
bool val = true;
while(val)
{
   cout << "\n<---------MENU----------->" << endl;
   cout << "This program is using random values from 0 to 1000" << endl;
   cout << "1. To calculate the sum without using CUDA" << endl;
   cout << "2. To calculate the sum using CUDA" << endl;
   cout << "3. For Exit" << endl;

   int ch; cout << "\nEnter your Choice: ";
   cin >> ch;

   switch(ch)
   {
   case 1:
      {
         double start = timeCal();
         sum(a, b, d, n);
         double end = timeCal();

         double time = end - start;

         cout << "Result Vector: ";
         for(int i = 0; i<n; i++)
         {
            cout << d[i] << " ";
         }
         cout << endl;

         cout << "Time taken for simple vector addition: " << time << " seconds\n";

         break;
      }

   case 2:
      {
         // Device vectors
         int *d_a, *d_b, *d_c;
         cudaMalloc(&d_a, n * sizeof(int));
         cudaMalloc(&d_b, n * sizeof(int));
         cudaMalloc(&d_c, n * sizeof(int));

         // Copy vectors to device
         cudaMemcpy(d_a, a.data(), n * sizeof(int), cudaMemcpyHostToDevice);
         cudaMemcpy(d_b, b.data(), n * sizeof(int), cudaMemcpyHostToDevice);

         double start = timeCal();
         // Define lambda function for the CUDA operation
         auto cuda_operation = [&]() {
            int blockSize = 256;
            int gridSize = (n + blockSize - 1) / blockSize;
            vectorAdd<<<gridSize, blockSize>>>(d_a, d_b, d_c, n);
            cudaDeviceSynchronize(); // Wait for the kernel to finish
         };
```

```
            // Measure time taken for CUDA operation
            cuda_operation();

            double end = timeCal();

            double time = end - start;

            // Copy result back to host
            copyResult(c, d_c, n);

            // Verify result
            cout << "Result of vector addition: \n";
            for (int i = 0; i < 10; i++) {
               cout << c[i] << " ";
            }
            cout << endl;

            cout << "Time taken for CUDA operation: " << time << "seconds\n";

            // Free device memory
            cudaFree(d_a);
            cudaFree(d_b);
            cudaFree(d_c);
            break;

         }

      case 3:
         {
            cout <<"Thank you!!" << endl;
            val = false;
            break;
         }
      default:
         {
            cout << "Invalid Choice, Please Try again!!"<< endl;
            cout << endl;
            break;
         }
      }
   }

   return 0;
}
```

Output:


rahul@akpc: ~/Desktop/HPC
rahul@akpc:~/Desktop/HPC$ nvcc -o as1 7446Assignment4a.cu
rahul@akpc:~/Desktop/HPC$ ./as1
Enter the size of the vectors: 10
<---------MENU----------->
This program is using random values from 0 to 1000
1. To calculate the sum without using CUDA
2. To calculate the sum using CUDA
3. For Exit
Enter your Choice: 1
Result Vector: 1269 1692 1128 878 1070 389 749 1689 966 908
Time taken for simple vector addition: 1.082e-06 seconds
<---------MENU----------->
This program is using random values from 0 to 1000
1. To calculate the sum without using CUDA
2. To calculate the sum using CUDA
3. For Exit
Enter your Choice: 2
Result of vector addition:
1269 1692 1128 878 1070 389 749 1689 966 908
Time taken for CUDA operation: 0.000127599seconds
<---------MENU----------->
This program is using random values from 0 to 1000
1. To calculate the sum without using CUDA
2. To calculate the sum using CUDA
3. For Exit
Enter your Choice: 3
Thank you!!
rahul@akpc:~/Desktop/HPC$

2. Matrix Multiplication using CUDA C.

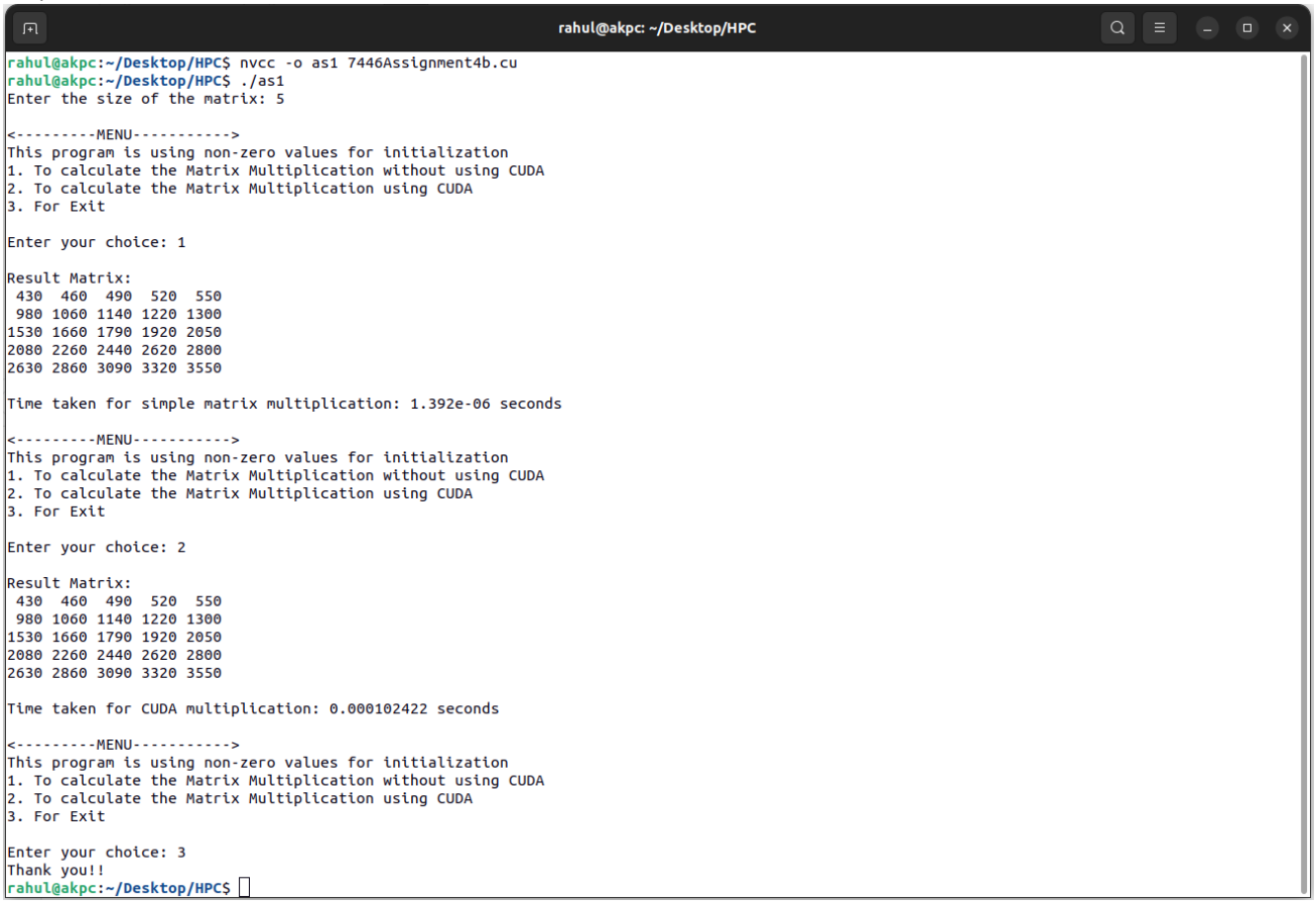
Code:

```
#include <bits/stdc++.h>

using namespace std;

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

// Host function to measure time
double timeCal() {
   return
chrono::duration_cast<chrono::duration<double>>(chrono::steady_clock::now().time_since_epoch()).count();
}

// Matrix multiplication without using CUDA
void matrixMul(vector<int>& a, vector<int>& b, vector<int>& d, int n) {
   for (int row = 0; row < n; ++row) {
      for (int col = 0; col < n; ++col) {
         int sum = 0;
         for (int k = 0; k < n; ++k) {
            sum += a[row * n + k] * b[k * n + col];
         }
         d[row * n + col] = sum;
      }
   }
}

// Function to print matrix
// Function to print matrix with aligned columns
void printMatrix(const vector<int>& matrix, int n) {
   int maxDigits = 0;
   for (int i = 0; i < n * n; ++i) {
      int digits = to_string(matrix[i]).length();
      if (digits > maxDigits) {
         maxDigits = digits;
      }
   }

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

```
        for (int j = 0; j < n; ++j) {
            cout << setw(maxDigits) << matrix[i * n + j] << " ";
        }
        cout << endl;
    }
}


int main() {
    int n;
    cout << "Enter the size of the matrix: ";
    cin >> n;

    vector<int> a(n * n);
    vector<int> b(n * n);
    vector<int> c(n * n);
    vector<int> d(n * n);

    // Initialize matrices with non-zero values
    for (int i = 0; i < n * n; ++i) {
        a[i] = i + 1; // Non-zero values
        b[i] = (i + 1) * 2; // Non-zero values
    }

    bool val = true;
    while (val) {
        cout << "\n<---------MENU----------->" << endl;
        cout << "This program is using non-zero values for initialization" << endl;
        cout << "1. To calculate the Matrix Multiplication without using CUDA" << endl;
        cout << "2. To calculate the Matrix Multiplication using CUDA" << endl;
        cout << "3. For Exit" << endl;

        int ch; cout << "\nEnter your choice: ";
        cin >> ch;

        switch (ch) {
            case 1: {
                double start = timeCal();
                // Perform matrix multiplication on host
                matrixMul(a, b, d, n);
                double end = timeCal();
                cout << "\nResult Matrix: " << endl;
                printMatrix(d, n);
                cout << "\nTime taken for simple matrix multiplication: " << end - start << " seconds\n";

                break;
            }

            case 2: {
                int *d_a, *d_b, *d_c;
                size_t bytes = n * n * sizeof(int);

                // Allocate memory on device
                cudaMalloc(&d_a, bytes);
```

```
            cudaMalloc(&d_b, bytes);
            cudaMalloc(&d_c, bytes);

            // Copy matrices from host to device
            cudaMemcpy(d_a, a.data(), bytes, cudaMemcpyHostToDevice);
            cudaMemcpy(d_b, b.data(), bytes, cudaMemcpyHostToDevice);

            // Launch kernel
            dim3 threadsPerBlock(16, 16);
            dim3 blocksPerGrid((n + threadsPerBlock.x - 1) / threadsPerBlock.x,
                        (n + threadsPerBlock.y - 1) / threadsPerBlock.y);

            double start = timeCal();
            matrixMul<<<blocksPerGrid, threadsPerBlock>>>(d_a, d_b, d_c, n);
            cudaDeviceSynchronize();
            double end = timeCal();


            // Copy result back to host
            cudaMemcpy(c.data(), d_c, bytes, cudaMemcpyDeviceToHost);

            cout << "\nResult Matrix: " << endl;
            printMatrix(c, n);

            cout << "\nTime taken for CUDA multiplication: " << end - start << " seconds\n";

            // Free device memory
            cudaFree(d_a);
            cudaFree(d_b);
            cudaFree(d_c);
            break;
          }

          case 3: {
            cout << "Thank you!!" << endl;
            val = false;
            break;
          }

          default: {
            cout << "Invalid Choice, Please Try again!!" << endl;
            break;
          }
        }
      }

      return 0;
    }
```

Output:


rahul@akpc: ~/Desktop/HPC
rahul@akpc:~/Desktop/HPC$ nvcc -o as1 7446Assignment4b.cu
rahul@akpc:~/Desktop/HPC$ ./as1
Enter the size of the matrix: 5
<---------MENU----------->
This program is using non-zero values for initialization
1. To calculate the Matrix Multiplication without using CUDA
2. To calculate the Matrix Multiplication using CUDA
3. For Exit
Enter your choice: 1
Result Matrix:
430 460 490 520 550
980 1060 1140 1220 1300
1530 1660 1790 1920 2050
2080 2260 2440 2620 2800
2630 2860 3090 3320 3550
Time taken for simple matrix multiplication: 1.392e-06 seconds
<---------MENU----------->
This program is using non-zero values for initialization
1. To calculate the Matrix Multiplication without using CUDA
2. To calculate the Matrix Multiplication using CUDA
3. For Exit
Enter your choice: 2
Result Matrix:
430 460 490 520 550
980 1060 1140 1220 1300
1530 1660 1790 1920 2050
2080 2260 2440 2620 2800
2630 2860 3090 3320 3550
Time taken for CUDA multiplication: 0.000102422 seconds
<---------MENU----------->
This program is using non-zero values for initialization
1. To calculate the Matrix Multiplication without using CUDA
2. To calculate the Matrix Multiplication using CUDA
3. For Exit
Enter your choice: 3
Thank you!!
rahul@akpc:~/Desktop/HPC$