



**VIT**<sup>®</sup>  
**UNIVERSITY**  
(Estd. u/s 3 of UGC Act 1956)

**FALL – SEMESTER**

**Course Code: MCSE503P**

**Course-Title: – Computer Architecture and Organization**

**DIGITAL ASSIGNMENT - V  
(LAB)**

**Name: Nidhi Singh**

**Reg. No:22MAI0015**

**Slot- L53+L54**

1. Write a c program to estimate the value of PI using open MP or MPI.

```
#include <iostream>
using namespace std;
void monteCarlo(int N, int K)
{
    double x, y;
    double d;
    int pCircle = 0;
    int pSquare = 0;
    int i = 0;
    #pragma omp parallel firstprivate(x, y, d, i) reduction(+ : pCircle, pSquare)
    num_threads(K)
    {
        srand48((int)time(NULL));
        for (i = 0; i < N; i++)
        {
            x = (double)drand48();
            y = (double)drand48();
            d = ((x * x) + (y * y));
            if (d <= 1)
            {
                pCircle++;
            }
            pSquare++;
        }
    }
    double pi = 4.0 * ((double)pCircle / (double)(pSquare));
    cout << "Final Estimation of Pi = " << pi;
}
int main()
{
    int N = 100000;
    int K = 8;
    monteCarlo(N, K);
}
```

**Output :-**

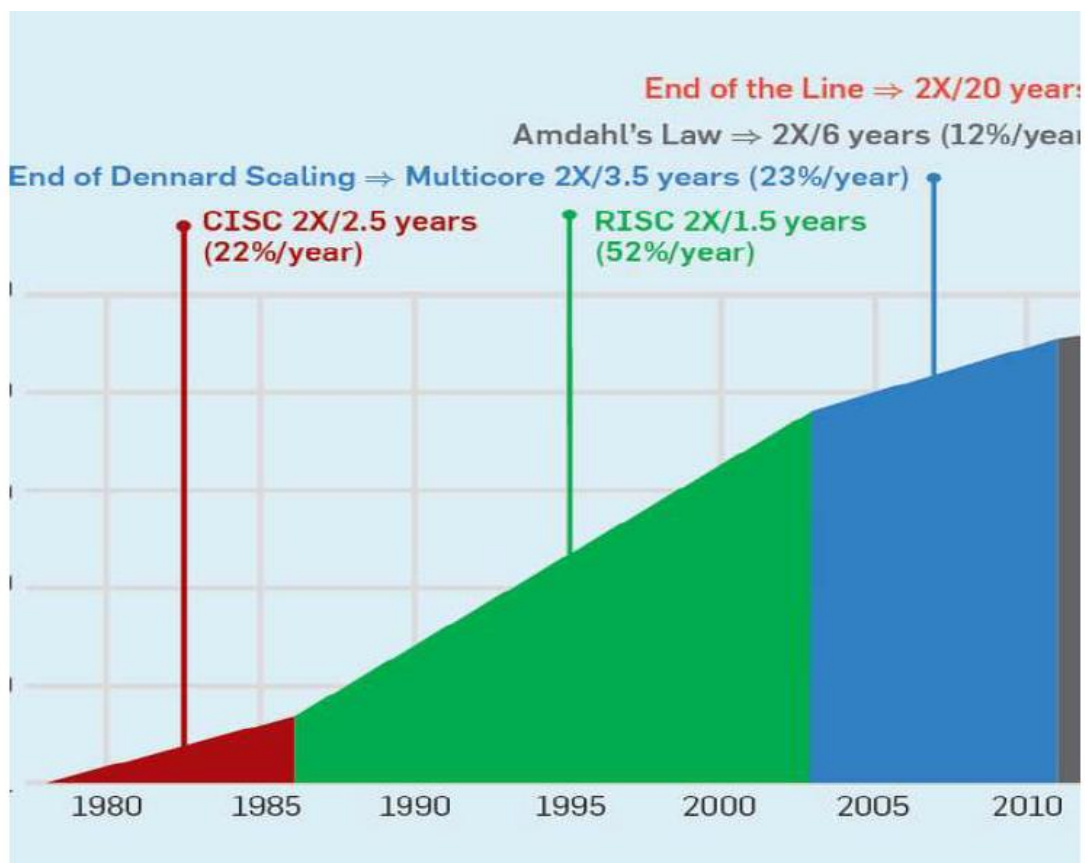
```
Final Estimation of Pi = 3.14256
```

**2.a. Study about GPU computing.****History of GPU Computing**

GPUs have historically been used to speed up memory-intensive computer graphics tasks like picture rendering and video decoding. These issues lend themselves well to parallelization. A GPU appeared to be an essential component of graphical processing because of its multiple cores and greater memory bandwidth rendering.

While GPU-driven parallel computing was necessary for graphical rendering, several scientific computing tasks also looked to benefit greatly from its use. As a result, GPU computing began to advance more quickly in 2006 and became appropriate for a variety of general-purpose computer activities.

A constant increase in GPU computing performance was made possible by the improvement of the existing GPU instruction sets and the expansion of the number of them that could be performed in a single clock cycle. As Moore's law has slowed today and some even claim it is no longer valid.



Huang's law extends Moore's law - the performance of GPUs will more than double every two years.

## CPU vs. GPU

While a CPU is latency-oriented and can handle complex linear tasks at speed, a GPU is throughput oriented, which allows for enormous parallelization. Architecturally, a CPU is composed of a few cores with lots of cache memory that can handle few software threads at the same time using sequential serial processing. In contrast, a GPU is composed of thousands smaller cores that can manage multiple threads simultaneously. Even though a CPU can handle a considerable number of tasks, it wouldn't be as fast as GPU doing so. A GPU breaks down complex problems into thousands of separate tasks and works through them simultaneously.

## GPU Computing Strengths & Weaknesses

A GPU is a specialized co-processor that excels at some tasks and is not so good at others. It works intandem with a CPU to increase the throughput of data and the number of concurrent calculations within the application.

### □ **Arithmetic Intensity:**

GPUs can cope extremely well with high arithmetic intensity. The algorithm is a good candidate for a GPU acceleration, if its ratio of math to memory operations is at least 10:1. If this is the case, your algorithm can benefit from the GPU's basic linear algebra subroutines (BLAS) and numerous arithmetic logic units (ALU).

### □ **High Degree of Parallelism:**

Parallel computing is a type of computation where many independent calculations are carried out simultaneously. Large problems can often be divided into smaller pieces which are then solved concurrently. GPU computing is designed to work like that. For instance, if it is possible to vectorize your data and adjust the algorithm to work on a set of values all at once, you can easily reap the benefits of GPU computing.

### □ **Sufficient GPU Memory:**

Ideally your data batch has to fit into the native memory of your GPU, in order to be processed seamlessly. Although there are workarounds to use multiple GPUs simultaneously or streamline your data from system memory, limited PCIe bandwidth may become a major performance bottleneck in such scenarios.

### □ **Enough Storage Bandwidth:**

In GPU computing you typically work with large amounts of data where storage bandwidth is crucial. Today the bottleneck for GPU-based scientific computing is no longer floating points per second (FLOPS), but I/O operations per second (IOPS). As a rule of thumb, it's always a good idea to evaluate your system's global bottleneck. If you find out that your GPU acceleration gains will be outweighed by the storage throughput limitations, optimize your storage solution first.

## GPU Computing Applications

GPU computing is being used for numerous real-world applications. Many prominent science and engineering fields that we take for granted today would have not progressed so fast, if not GPU computing.

### □ **Deep Learning**

Deep learning is a subset of machine learning. Its implementation is based on artificial neural networks. Essentially, it mimics the brain, having neuron layers work in parallel. Since data is represented as a set of vectors, deep learning is well-suited for GPU computing. You can easily experience up to 4x performance gains when training your convolutional neural network on a Dedicated Server with a GPU accelerator. As a cherry on top, every major deep learning framework like TensorFlow and PyTorch already allows you to use GPU computing out-of-thebox with no code changes.

### □ **Drug Design**

The successful discovery of new drugs is hard in every respect. We have all become aware of this during the Covid-19 pandemic. Eroom's law states that the cost of discovering a new drug roughly doubles every nine years. Modern

GPU computing aims to shift the trajectory of Eroom's law. Nvidia is currently building Cambridge-1 - the most powerful supercomputer in the UK - dedicated to AI research in healthcare and drug design.

#### □ **Seismic Imaging**

Seismic imaging is used to provide the oil and gas industry with knowledge of Earth's subsurface structure and detect oil reservoirs. The algorithms used in seismic data processing are evolving rapidly, so there's a huge demand for additional computing power. For instance, the Reverse Time Migration method can be accelerated up to 14 times when using GPU computing.

#### □ **Automotive design**

Flow field computations for transient and turbulent flow problems are highly compute-intensive and time-consuming. Traditional techniques often compromise on the underlying physics and are not very efficient. A new paradigm for computing fluid flows relies on GPU computing that can help achieve significant speed-ups over a single CPU, even up to a factor of 100

#### □ **Astrophysics**

GPU has dramatically changed the landscape of high-performance computing in astronomy. Take an N-body simulation for instance, that numerically approximates the evolution of a system of bodies in which each body continuously interacts with every other body. You can accelerate the all-pairs N-body algorithm up to 25 times by using GPU computing rather than using a highly tuned serial CPU implementation.

#### □ **Options pricing**

The goal of option pricing theory is to provide traders with an option's fair value that can then be incorporated into their trading strategies. Some type of Monte Carlo algorithm is often used in such simulations. GPU computing can help you achieve 27 times better performance per dollar compared to CPU-only approach.

#### □ **Weather forecasting**

Weather forecasting has greatly benefited from exponential growth of mere computing power in recent decades, but this free ride is nearly over. Today weather forecasting is being driven by fine-grained parallelism that is based on extensive GPU computing. This approach alone can ensure 20 times faster weather forecasting models.

#### □ **GPU Computing in the Cloud**

Even though GPU computing was once primarily associated with graphical rendering, it has grown into the main driving force of high-performance computing in many different scientific and engineering fields.

□ Most of the GPU computing work is now being done in the cloud or by using in-house GPU computing clusters. Here at Cherry Servers, we are offering dedicated GPU Servers with highend Nvidia GPU accelerators. Our infrastructure services can be used on-demand, which makes GPU computing easy and cost-effective.

□ Cloud vendors have democratized GPU computing, making it accessible for small and medium businesses world-wide. If Huang's law lasts, the performance of GPU will more than double every two years, and innovation will continue to sprout.

## **2.b. Write a C program using CUDA to find the matrix addition, subtraction and multiplication.**

### **Program:**

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#define BLOCK_SIZE 16
__global__ void gpu_matrix_ops(int* a, int* b, int* c, int* d, int* e, int m, int
n, int k)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
```

```

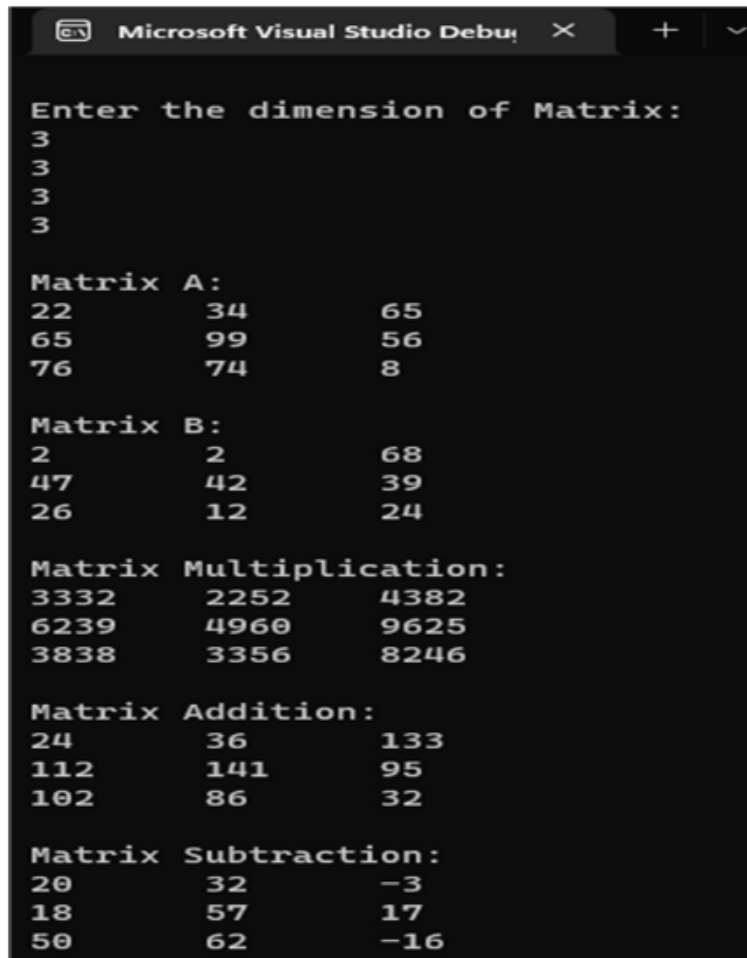
int sum = 0;
if (col < k && row < m)
{
for (int i = 0; i < n; i++)
{
sum += a[row * n + i] * b[i * k + col];
}
c[row * k + col] = sum; //Multiplication
d[row * k + col] = a[row * k + col] + b[row * k + col]; //Addition
e[row * k + col] = a[row * k + col] - b[row * k + col]; //Subtraction
}
}

int main(int argc, char const* argv[])
{
int m, n, k;
srand(3333);
printf("\nEnter the dimension of Matrix:\n");
scanf("%d %d %d\n", &m, &n, &k);
int* h_a, * h_b, * h_c, * h_d, * h_e;
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_d, sizeof(int) * m * k);
cudaMallocHost((void**)&h_e, sizeof(int) * m * k);
printf("\nMatrix A:\n");
for (int i = 0; i < m; ++i) {
for (int j = 0; j < n; ++j) {
h_a[i * n + j] = rand() % 100;
printf("%d\t", h_a[i * n + j]);
}
printf("\n");
}
printf("\nMatrix B:\n");
for (int i = 0; i < n; ++i) {
for (int j = 0; j < k; ++j) {
h_b[i * k + j] = rand() % 100;
printf("%d\t", h_b[i * k + j]);
}
printf("\n");
}
int* d_a, * d_b, * d_c, * d_d, * d_e;
cudaMalloc((void**)&d_a, sizeof(int) * m * n);
cudaMalloc((void**)&d_b, sizeof(int) * n * k);
cudaMalloc((void**)&d_c, sizeof(int) * m * k);
cudaMalloc((void**)&d_d, sizeof(int) * m * k);
cudaMalloc((void**)&d_e, sizeof(int) * m * k);
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);
gpu_matrix_ops << <dimGrid, dimBlock >> > (d_a, d_b, d_c, d_d, d_e, m, n, k);
cudaMemcpy(h_c, d_c, sizeof(int) * m * k, cudaMemcpyDeviceToHost);
cudaMemcpy(h_d, d_d, sizeof(int) * m * k, cudaMemcpyDeviceToHost);
cudaMemcpy(h_e, d_e, sizeof(int) * m * k, cudaMemcpyDeviceToHost);
cudaThreadSynchronize();
printf("\nMatrix Multiplication:\n");
for (int i = 0; i < m; ++i)
{
    for (int j = 0; j < k; ++j)
    {
        printf("%d\t", h_c[i * n + j]);
    }
    printf("\n");
}
printf("\nMatrix Addition:\n");
for (int i = 0; i < m; ++i)
{
    for (int j = 0; j < k; ++j)
    {
        printf("%d\t", h_d[i * n + j]);
    }
    printf("\n");
}
printf("\nMatrix Subtraction:\n");
for (int i = 0; i < m; ++i)
{
    for (int j = 0; j < k; ++j)
    {
        printf("%d\t", h_e[i * n + j]);
    }
    printf("\n");
}
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
cudaFree(d_d);
cudaFree(d_e);
cudaFreeHost(h_a);
cudaFreeHost(h_b);
cudaFreeHost(h_c);
cudaFreeHost(h_d);
cudaFreeHost(h_e);
return 0;
}
```

Output :-



```
Microsoft Visual Studio Debug Console
Enter the dimension of Matrix:
3
3
3
3

Matrix A:
22      34      65
65      99      56
76      74      8

Matrix B:
2       2       68
47      42      39
26      12      24

Matrix Multiplication:
3332    2252    4382
6239    4960    9625
3838    3356    8246

Matrix Addition:
24      36      133
112     141     95
102     86      32

Matrix Subtraction:
20      32      -3
18      57      17
50      62     -16
```