410255:LP-V. HPC	
Experiment No: 4	CUDA Programming

Aim : Write a CUDA Program for :

1. Addition of two large vectors

2. Matrix Multiplication using CUDA C

Theory :

1. **Addition of Two Large Vectors**

```
#include<iostream>
                    #include<cstdlib>
                    using namespace std;
                    //VectorAdd parallel function
                    __global__ void vectorAdd(int *a, int *b, int *result, int n)
                    {
                       int tid=threadIdx.x+blockIdx.x*blockDim.x;
                       if(tid<n)
                       {
                          result[tid]=a[tid]+b[tid];
                       }
                    }
                    int main()
                    {
                       int *a,*b,*c;
                       int *a_dev,*b_dev,*c_dev;
                       int n=1<<24;

                       a=new int[n];
                       b=new int[n];
                       c=new int[n];
                       int *d=new int[n];
                       int size=n*sizeof(int);
                       cudaMalloc(&a_dev,size);
                       cudaMalloc(&b_dev,size);
                       cudaMalloc(&c_dev,size);

                       //Array initialization..You can use Randon function to assign values
                       for(int i=0;i<n;i++)
                       {
                          a[i]=1;
                          b[i]=2;
                          d[i]=a[i]+b[i]; //calculating serial addition
```

```
    }


    cudaEvent_t start,end;

    cudaEventCreate(&start);
    cudaEventCreate(&end);

    cudaMemcpy(a_dev,a,size,cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev,b,size,cudaMemcpyHostToDevice);
    int threads=1024;
    int blocks=(n+threads-1)/threads;
    cudaEventRecord(start);

    //Parallel addition program
    vectorAdd<<<blocks,threads>>>(a_dev,b_dev,c_dev,n);

    cudaEventRecord(end);
    cudaEventSynchronize(end);

    float time=0.0;
    cudaEventElapsedTime(&time,start,end);

    cudaMemcpy(c,c_dev,size,cudaMemcpyDeviceToHost);

    //Calculate the error term.
    int error=0;
    for(int i=0;i<n;i++){
       error+=d[i]-c[i];
       //cout<<" gpu "<<c[i]<<" CPU "<<d[i];
    }

    cout<<"Error : "<<error;
    cout<<"\nTime Elapsed:  "<<time;

    return 0;
}
```

2. Matrix Multiplication using CUDA C

```
#include<iostream
>
                #include<cstdlib>
                #include<cmath>
                using namespace std;
                //Matrix multiplication Cuda
                __global__ void matrixMultiplication(int *a, int *b, int *c, int n)
                {
                   int row=threadIdx.y+blockDim.y*blockIdx.y;
                   int col=threadIdx.x+blockDim.x*blockIdx.x;
                   int sum=0;

                   if(row<n && col<n)
                   for(int j=0;j<n;j++)
                   {
                      sum=sum+a[row*n+j]*b[j*n+col];
                   }

                   c[n*row+col]=sum;
                }
                int main()
                {
                   int *a,*b,*c;
                   int *a_dev,*b_dev,*c_dev;
                   int n=3;

                   a=new int[n*n];
                   b=new int[n*n];
                   c=new int[n*n];
                   int *d=new int[n*n];
                   int size=n*n*sizeof(int);
                   cudaMalloc(&a_dev,size);
                   cudaMalloc(&b_dev,size);
                   cudaMalloc(&c_dev,size);

                   //Array initialization
                   for(int i=0;i<n*n;i++)
                   {
                        a[i]=2;  //rand()%n;
                      b[i]=1;//rand()%n;
                     // d[i]=a[i]+b[i];
                   }
```

```
    cudaEvent_t start,end;

    cudaEventCreate(&start);
    cudaEventCreate(&end);

    cudaMemcpy(a_dev,a,size,cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev,b,size,cudaMemcpyHostToDevice);


    dim3 threadsPerBlock(n, n);
    dim3 blocksPerGrid(1, 1);

    if(n*n>512){
       threadsPerBlock.x=512;
       threadsPerBlock.y=512;
       blocksPerGrid.x=ceil((double)n/(double)threadsPerBlock.x);
       blocksPerGrid.y=ceil((double)n/(double)threadsPerBlock.y);
    }
    //GPU Multiplication
    cudaEventRecord(start);

matrixMultiplication<<<blocksPerGrid,threadsPerBlock>>>(a_dev,b_dev,c_d
ev,n);

    cudaEventRecord(end);
    cudaEventSynchronize(end);

    float time=0.0;
    cudaEventElapsedTime(&time,start,end);

    cudaMemcpy(c,c_dev,size,cudaMemcpyDeviceToHost);


    //CPU matrix multiplication
    int sum=0;
    for(int row=0;row<n;row++)
    {
       for(int col=0;col<n;col++)
       {
         sum=0;
         for(int k=0;k<n;k++)
            sum=sum+a[row*n+k]*b[k*n+col];
          d[row*n+col]=sum;
       }

    }
```

```
    int error=0;
    for(int i=0;i<n*n;i++){
        error+=d[i]-c[i];
        //cout<<" gpu "<<c[i]<<" CPU "<<d[i]<<endl;
    }

    cout<<"Error : "<<error;
    cout<<"\nTime Elapsed:  "<<time;

    return 0;
}
```