Name: Arppit Madankar
Roll no: 82
Department: IT
Subject: Parallel Computing Lab
Registration No: 18010822

Practical 7

Aim: Write a CUDA Program to perform vector addition.

Theory:

What is CUDA?

CUDA is a parallel computing platform and programming model developed by Nvidia for general computing on its own GPUs (graphics processing units). CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for the parallelizable part of the computation.

Why you need CUDA?

CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for any parallelizable part of a computation. In contrast to prior GPU programming interfaces such as Direct3D and OpenGL, which required advanced skills in graphics programming, CUDA makes it a lot easier for developers and software engineers to implement parallel programming since it is compatible with many familiar programming languages such as C, C++ or Fortran. The developer only needs to add extensions of these languages in the form of a few basic keywords, which gives them direct access to the GPU's virtual instruction set and parallel computational elements for the execution of compute kernels.

CUDA performance boost:

CUDA has improved and broadened its scope over the years, more or less in lockstep with improved NVidia GPUs. As of CUDA version 9.2, using multiple P100 server GPUs, you can realize up to 50x performance improvements over CPUs. The V100 (not shown in this figure) is another 3x faster for some loads. The previous generation of server GPUs, the

K80, offered 5x to 12x performance improvements over CPUs.

Program for Vector Addition:

```
%%cu
#include <iostream>
#include <cuda.h>
using namespace std;
int *a, *b; // host data
int *c, *c2; // results
__global__ void vecAdd(int *A,int *B,int *C,int N)
{
 int i = blockIdx.x * blockDim.x + threadIdx.x;
 C[i] = A[i] + B[i];
}
void vecAdd_h(int *A1,int *B1, int *C1, int N)
{
 for(int i=0;i<N;i++)
 C1[i] = A1[i] * B1[i];
}
int main(int argc,char **argv)
{
 printf("Begin \n");
 int n=10000000;
 int nBytes = n*sizeof(int);
 int block_size, block_no;
  a = (int *)malloc(nBytes);
 b = (int *)malloc(nBytes);
 c = (int *)malloc(nBytes);
 c2 = (int *)malloc(nBytes);
  int *a_d,*b_d,*c_d;
 block_size=4000;
 block_no = n/block_size;
 dim3 dimBlock(block_size,1,1);
 dim3 dimGrid(block_no,1,1);
 for(int i=0;i<n;i++)
 a[i]=i,b[i]=i;
 printf("Allocating device memory on host..\n");
  cudaMalloc((void **)&a_d,n*sizeof(int));
 cudaMalloc((void **)&b_d,n*sizeof(int));
 cudaMalloc((void **)&c_d,n*sizeof(int));
 printf("Copying to device..\n");

 cudaMemcpy(a_d,a,n*sizeof(int),cudaMemcpyHostToDevice); cudaMemcpy(b_d,b,n*sizeof(int),cudaMemcpyHostT
oDevice); clock_t start_d=clock();

 printf("Doing GPU Vector add\n");
 vecAdd<<<block_no,block_size>>>(a_d,b_d,c_d,n);
```

```
cudaThreadSynchronize();
clock_t end_d = clock();
clock_t start_h = clock();
 printf("Doing CPU Vector add\n");
```



```
vecAdd_h(a,b,c2,n);
clock_t end_h = clock();
double time_d = (double)(end_d-start_d)/CLOCKS_PER_SEC;
double time_h = (double)(end_h-start_h)/CLOCKS_PER_SEC;
cudaMemcpy(c,c_d,n*sizeof(int),cudaMemcpyDeviceToHost);
 printf("N = %d \nTime of GPU = %f \nTime of CPU = %f\n",n,time_d,time_h);
cudaFree(a_d);
cudaFree(b_d);
cudaFree(c_d);
return 0;
}
```

Output:

Conclusion: Thus, we studied CUDA and successfully implemented and understood the CUDA program to perform vector addition.