Practical No:4

```
//CUDA Program for Addition of Two Large Vectors:
#include <stdio.h>
#include <stdlib.h>
 #include <math.h>
// CUDA kernel. Each thread takes care of one element of C
_global void vecAdd(double a, double "b, double *c, int n)
{
// Get our global thread ID int id blockTdx.x'blockDim.x+threadTdx.x;
// Make sure we do not go out of bounds.
if (id n)
c[id] a[id]+b[id];
}
int main(int argc, char argv[])
{
// Size of of vectors int n = 100000;
//Host input vectors.
double tha;
double 'n b;
//Host output vector
double hc;
// Device input vectors
double 'd_a;
double d b;
//Device output vector
double d c;
Size, in bytes, of each vector size t bytes n'sizeof(double);
// Allocate memory for each vector on host h_a (double*)malloc(bytes); h_b (double)malloc(bytes);
hc (double*)malloc(bytes);
// Allocate memory for each vector on GPU cudaMalloc(&d_a, bytes);
cudaMalloc(&d_b, bytes);
```

```
cudaMalloc(&d.c, bytes);

int i;

// Initialize vectors on host

for(18; 1< n; i++) ( h_a[i] sin(1)*sin(i);

h_b[1] cos(1)cos(1);

}

// Copy host vectors to device cudaMemcpy(da, ha, bytes, cudaMemcpyHostToDevice);
cudaMemcpy(db, hb, bytes, cudaMemcpyHostToDevice);

int blockSize, gridSize;

// Number of threads in each thread block blockSize 1024;

//Number of thread blocks in grid gridsize (int)ceil((float)n/blocksize);

// Execute the kernel

vecAdd<<<gridSize, blockSize>>>(d_a, db, dc, n);

// Copy array back to host cudaMemcpy(hc, dc, bytes, cudaMemcpyDeviceTollost);

// Sum up vector c and print result divided by n, this should equal 1 within error

Double sum = 0; for(i=0; i<n; i++)

Sum + hoc[1];

Printf("final result: %f\n", sum/n);

// Release device memory cudaFree (d_a); cudaFree(db);

cudaFree(d_c);

// Release host memory free(ha);

Free(hb); free(h_c);

Return 0;

}
```

Output:

Final result: 0,499950