1. **Write a CUDA program for adding two 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 = blockIdx.x\*blockDim.x+threadIdx.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 vectors

int n = 10;

// Host input vectors

double \*h\_a;

double \*h\_b;

//Host output vector

double \*h\_c;

// 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);

h\_c = (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( i = 0; i < n; i++ ) {

h\_a[i] = i;

h\_b[i] = i;

}

// Copy host vectors to device

cudaMemcpy( d\_a, h\_a, bytes, cudaMemcpyHostToDevice);

cudaMemcpy( d\_b, h\_b, bytes, cudaMemcpyHostToDevice);

int blockSize, gridSize;

// Number of threads in each thread block

blockSize = 100;

// Number of thread blocks in grid

gridSize = (int)ceil((float)n/blockSize);

// Execute the kernel

vecAdd<<<gridSize, blockSize>>>(d\_a, d\_b, d\_c, n);

// Copy array back to host

cudaMemcpy( h\_c, d\_c, bytes, cudaMemcpyDeviceToHost );

// 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++)

printf(" %f + %f =%f\n",h\_a[i],h\_b[i],h\_c[i]);

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

// Release device memory

cudaFree(d\_a);

cudaFree(d\_b);

cudaFree(d\_c);

// Release host memory

free(h\_a);

free(h\_b);

free(h\_c);

return 0;

}