Lab-5 Programs on arrays in CUDA

Vector Addition

```
// Title  : Vector Addition using CUDA
// Author : Aditya Sinha
// Date   : 13/02/2026

#include <cuda_runtime.h>
#include <stdio.h>
#include <stdlib.h>

static const int threadsPerBlock = 256;

__global__ void vectorAdd(const int *a, const int *b, int *c, int n) {
    const int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        c[idx] = a[idx] + b[idx];
    }
}

int main(void) {
    int lenA = 0;
    int lenB = 0;
    scanf("%d", &lenA);
    scanf("%d", &lenB);

    if (lenA != lenB) {
        printf("Vector size mismatch: %d vs %d\n", lenA, lenB);
        return 1;
    }

    const int n = lenA;
    const int bytes = n * (int)sizeof(int);

    int *h_a = (int *)malloc(bytes);
    int *h_b = (int *)malloc(bytes);
    int *h_c = (int *)malloc(bytes);

    for (int i = 0; i < n; ++i) {
        h_a[i] = i;
        h_b[i] = n - i;
    }

    int *d_a = NULL;
    int *d_b = NULL;
    int *d_c = NULL;
    cudaMalloc((void **)&d_a, bytes);
    cudaMalloc((void **)&d_b, bytes);
    cudaMalloc((void **)&d_c, bytes);
    cudaMemcpy(d_a, h_a, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, bytes, cudaMemcpyHostToDevice);

    const int blocks = (n + threadsPerBlock - 1) / threadsPerBlock;
    vectorAdd<<<blocks, threadsPerBlock>>>(d_a, d_b, d_c, n);
    cudaMemcpy(h_c, d_c, bytes, cudaMemcpyDeviceToHost);

    for (int i = 0; i < n; ++i) {
        printf("%d ", h_c[i]);
    }
    printf("\n");

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

Q1.1 – Launch one thread per block for each element

```
vectorAdd<<<n, 1>>>(d_a, d_b, d_c, n);
```

```
Vector length: 4
Threads per block: 1
Blocks launched: 4
0 3 6 9
```

Q1.2 – Launch a single block with n threads

```
vectorAdd<<<1, n>>>(d_a, d_b, d_c, n);
```

```
Vector length: 4
Threads per block: 4
Blocks launched: 1
0 3 6 9
```

Q2 – Fix 256 threads per block and scale blocks for N

```
vectorAdd<<<(n + 255) / 256, 256>>>(d_a, d_b, d_c, n);
```

```
Vector length: 8
Threads per block: 256
Blocks launched: 1
0 3 6 9 12 15 18 21
```

Q3 – Compute sine values of input angles

```
// Title  : Radians to Sine using CUDA
// Author : Aditya Sinha
// Date   : 13/02/2026

#include <cuda_runtime.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

__global__ void radiansToSine(float *angles, float *sines, int n) {
    const int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        sines[idx] = sinf(angles[idx]);
    }
}

int main(void) {
    int n = 0;
    scanf("%d", &n);

    float *h_angles = (float *)malloc(n * sizeof(float));
    float *h_sines = (float *)malloc(n * sizeof(float));
    for (int i = 0; i < n; ++i) {
        scanf("%f", &h_angles[i]);
    }

    float *d_angles = NULL;
    float *d_sines = NULL;
    cudaMalloc((void **)&d_angles, n * sizeof(float));
    cudaMalloc((void **)&d_sines, n * sizeof(float));
    cudaMemcpy(d_angles, h_angles, n * sizeof(float),
cudaMemcpyHostToDevice);

    radiansToSine<<<(n + 255) / 256, 256>>>(d_angles, d_sines, n);
    cudaMemcpy(h_sines, d_sines, n * sizeof(float),
cudaMemcpyDeviceToHost);

    for (int i = 0; i < n; ++i) {
        printf("%f ", h_sines[i]);
    }
    printf("\n");

    cudaFree(d_angles);
    cudaFree(d_sines);
    free(h_angles);
    free(h_sines);

    return 0;
}
```

```
0.000000 0.707107 1.000000 -0.000000
```