Lab 5 – Programs on arrays in CUDA

Solved.cu

```
#include <stdio.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

__global__ void add(int *a, int *b, int *c) {
    int i = threadIdx.x;
    c[i] = a[i] + b[i];
}

int main(void) {
    int a, b, c;
    int *d_a, *d_b, *d_c;
    int size = sizeof(int);

    cudaMalloc((void**)&d_a, size);
    cudaMalloc((void**)&d_b, size);
    cudaMalloc((void**)&d_c, size);

    a = 3;
    b = 5;

    cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);

    add<<<1, 1>>>(d_a, d_b, d_c);

    cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);

    printf("Result: %d\n", c);

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    return 0;
}
```

Output:

```
6CSEC1@debian:~/Documents/230905152_PPL/L5$ nvcc -o abc solved.cu
6CSEC1@debian:~/Documents/230905152_PPL/L5$ ./abc
Result: 80
```

Q1.cu

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "device_launch_parameters.h"

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

int main() {
    int n;
    printf("Enter N: ");
    if (scanf("%d", &n) != 1) return 0;

    int *h_a = (int *)malloc(n * sizeof(int));
    int *h_b = (int *)malloc(n * sizeof(int));
    int *h_c = (int *)malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        printf("Enter a[%d]: ", i);
        scanf("%d", &h_a[i]);
    }
    for (int i = 0; i < n; i++) {
        printf("Enter b[%d]: ", i);
        scanf("%d", &h_b[i]);
    }

    int *d_a, *d_b, *d_c;
    cudaMalloc((void **)&d_a, n * sizeof(int));
    cudaMalloc((void **)&d_b, n * sizeof(int));
    cudaMalloc((void **)&d_c, n * sizeof(int));

    cudaMemcpy(d_a, h_a, n * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, n * sizeof(int), cudaMemcpyHostToDevice);

    // a) block size as N (one block with N threads)
```

```
    vecAdd<<<1, n>>>(d_a, d_b, d_c, n);
    cudaMemcpy(h_c, d_c, n * sizeof(int), cudaMemcpyDeviceToHost);
    printf("A) ");
    for (int i = 0; i < n; i++) printf("%d ", h_c[i]);
    printf("\n");

    // b) N threads (N blocks of 1 thread)
    vecAdd<<<n, 1>>>(d_a, d_b, d_c, n);
    cudaMemcpy(h_c, d_c, n * sizeof(int), cudaMemcpyDeviceToHost);
    printf("B) ");
    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;
}
```

Output:

```
6CSEC1@debian:~/Documents/230905152_PPL/L5$ nvcc -o abc q1.cu
6CSEC1@debian:~/Documents/230905152_PPL/L5$ ./abc
Enter N: 4
Enter a[0]: 11
Enter a[1]: 22
Enter a[2]: 33
Enter a[3]: 44
Enter b[0]: 55
Enter b[1]: 66
Enter b[2]: 77
Enter b[3]: 88
A) 66 88 110 132
B) 66 88 110 132
```

Q2.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "device_launch_parameters.h"
```

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

int main() {
    int n;
    printf("Enter N: ");
    if (scanf("%d", &n) != 1) return 0;

    int *h_a = (int *)malloc(n * sizeof(int));
    int *h_b = (int *)malloc(n * sizeof(int));
    int *h_c = (int *)malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        printf("Enter a[%d]: ", i);
        scanf("%d", &h_a[i]);
    }
    for (int i = 0; i < n; i++) {
        printf("Enter b[%d]: ", i);
        scanf("%d", &h_b[i]);
    }

    int *d_a, *d_b, *d_c;
    cudaMalloc((void **)&d_a, n * sizeof(int));
    cudaMalloc((void **)&d_b, n * sizeof(int));
    cudaMalloc((void **)&d_c, n * sizeof(int));

    cudaMemcpy(d_a, h_a, n * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, n * sizeof(int), cudaMemcpyHostToDevice);

    int threads = 256;
    int blocks = (n + threads - 1) / threads;
    vecAdd<<<blocks, threads>>>(d_a, d_b, d_c, n);

    cudaMemcpy(h_c, d_c, n * sizeof(int), 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;
}
```

Output:

```
6CSEC1@debian:~/Documents/230905152_PPL/L5$ nvcc -o abc q2.cu
6CSEC1@debian:~/Documents/230905152_PPL/L5$ ./abc
Enter N: 5
Enter a[0]: 10
Enter a[1]: 20
Enter a[2]: 30
Enter a[3]: 40
Enter a[4]: 50
Enter b[0]: 5
Enter b[1]: 15
Enter b[2]: 25
Enter b[3]: 35
Enter b[4]: 45
15 35 55 75 95
```

Q3.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "device_launch_parameters.h"
#include <math.h>

__global__ void angleSign(const float *in, int *out, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        float v = in[i];
        out[i] = (v > 0.0f) - (v < 0.0f);
    }
}

int main() {
    int n;
    printf("Enter N: ");
    if (scanf("%d", &n) != 1) return 0;
```

```
    float *h_in = (float *)malloc(n * sizeof(float));
    int *h_out = (int *)malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        printf("Enter angle[%d] (radians): ", i);
        scanf("%f", &h_in[i]);
    }

    float *d_in;
    int *d_out;
    cudaMalloc((void **)&d_in, n * sizeof(float));
    cudaMalloc((void **)&d_out, n * sizeof(int));

    cudaMemcpy(d_in, h_in, n * sizeof(float), cudaMemcpyHostToDevice);

    int threads = 256;
    int blocks = (n + threads - 1) / threads;
    angleSign<<<blocks, threads>>>(d_in, d_out, n);

    cudaMemcpy(h_out, d_out, n * sizeof(int), cudaMemcpyDeviceToHost);
    for (int i = 0; i < n; i++) printf("%d ", h_out[i]);
    printf("\n");

    cudaFree(d_in);
    cudaFree(d_out);
    free(h_in);
    free(h_out);
    return 0;
}
```

Output:

```
6CSEC1@debian:~/Documents/230905152_PPL/L5$ nvcc -o abc q3.cu
6CSEC1@debian:~/Documents/230905152_PPL/L5$ ./abc
Enter N: 4
Enter angle[0] (radians): 0
Enter angle[1] (radians): 1.5
Enter angle[2] (radians): 3.14159
Enter angle[3] (radians): 4.71239
0 1 1 1
```

A1.cu

```
#include <stdio.h>
```

```cpp
#include <stdlib.h>
#include <cuda_runtime.h>
#include "device_launch_parameters.h"

__global__ void saxpy(float a, const float *x, float *y, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        y[i] = a * x[i] + y[i];
    }
}

int main() {
    int n;
    float a;
    printf("Enter N: ");
    if (scanf("%d", &n) != 1) return 0;
    printf("Enter a: ");
    if (scanf("%f", &a) != 1) return 0;

    float *h_x = (float *)malloc(n * sizeof(float));
    float *h_y = (float *)malloc(n * sizeof(float));

    for (int i = 0; i < n; i++) {
        printf("Enter x[%d]: ", i);
        scanf("%f", &h_x[i]);
    }
    for (int i = 0; i < n; i++) {
        printf("Enter y[%d]: ", i);
        scanf("%f", &h_y[i]);
    }

    float *d_x, *d_y;
    cudaMalloc((void **)&d_x, n * sizeof(float));
    cudaMalloc((void **)&d_y, n * sizeof(float));

    cudaMemcpy(d_x, h_x, n * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, h_y, n * sizeof(float), cudaMemcpyHostToDevice);

    int threads = 256;
    int blocks = (n + threads - 1) / threads;

    saxpy<<<blocks, threads>>>(a, d_x, d_y, n);
```

```
    cudaMemcpy(h_y, d_y, n * sizeof(float), cudaMemcpyDeviceToHost);
    for (int i = 0; i < n; i++) {
        printf("%.2f ", h_y[i]);
    }
    printf("\n");
    cudaFree(d_x);
    cudaFree(d_y);
    free(h_x);
    free(h_y);

    return 0;
}
```

Output:

```
6CSEC1@debian:~/Documents/230905152_PPL/L5$ nvcc -o abc a1.cu
6CSEC1@debian:~/Documents/230905152_PPL/L5$ ./abc
Enter N: 3
Enter a: 2
Enter x[0]: 1
Enter x[1]: 2
Enter x[2]: 3
Enter y[0]: 4
Enter y[1]: 5
Enter y[2]: 6
6.00 9.00 12.00
```

A2.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "device_launch_parameters.h"

__global__ void sortRows(float *mat, int rows, int cols) {
    int r = blockIdx.x * blockDim.x + threadIdx.x;
    if (r >= rows) return;

    int base = r * cols;
    for (int i = 0; i < cols - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < cols; j++) {
            if (mat[base + j] < mat[base + minIdx]) {
                minIdx = j;
            }
```

```
        }
        if (minIdx != i) {
            float tmp = mat[base + i];
            mat[base + i] = mat[base + minIdx];
            mat[base + minIdx] = tmp;
        }
    }
}

int main() {
    int rows, cols;
    printf("Enter rows and cols: ");
    if (scanf("%d %d", &rows, &cols) != 2) return 0;

    int size = rows * cols;
    float *h_mat = (float *)malloc(size * sizeof(float));

    for (int i = 0; i < size; i++) {
        printf("Enter element[%d]: ", i);
        scanf("%f", &h_mat[i]);
    }

    float *d_mat;
    cudaMalloc((void **)&d_mat, size * sizeof(float));
    cudaMemcpy(d_mat, h_mat, size * sizeof(float), cudaMemcpyHostToDevice);

    int threads = 256;
    int blocks = (rows + threads - 1) / threads;
    sortRows<<<blocks, threads>>>(d_mat, rows, cols);

    cudaMemcpy(h_mat, d_mat, size * sizeof(float), cudaMemcpyDeviceToHost);

    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            printf("%.2f ", h_mat[r * cols + c]);
        }
        printf("\n");
    }

    cudaFree(d_mat);
    free(h_mat);
    return 0;
}
```

Output:

```
6CSEC1@debian:~/Documents/230905152_PPL/L5$ nvcc -o abc a2.cu
6CSEC1@debian:~/Documents/230905152_PPL/L5$ ./abc
Enter rows and cols: 3 4
Enter element[0]: 9
Enter element[1]: 3
Enter element[2]: 7
Enter element[3]: 1
Enter element[4]: 8
Enter element[5]: 2
Enter element[6]: 6
Enter element[7]: 4
Enter element[8]: 5
Enter element[9]: 10
Enter element[10]: 3
Enter element[11]: 12
1.00 3.00 7.00 9.00
2.00 4.00 6.00 8.00
3.00 5.00 10.00 12.00
```

A3.cu

```cuda
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "device_launch_parameters.h"

__global__ void oddEvenStep(int *arr, int n, int phase) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    int i = 2 * tid + (phase & 1);

    if (i + 1 < n) {
        if (arr[i] > arr[i + 1]) {
            int tmp = arr[i];
            arr[i] = arr[i + 1];
            arr[i + 1] = tmp;
        }
    }
}

int main() {
    int n;
```

```
    printf("Enter N: ");
    if (scanf("%d", &n) != 1) return 0;

    int *h_arr = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        printf("Enter arr[%d]: ", i);
        scanf("%d", &h_arr[i]);
    }

    int *d_arr;
    cudaMalloc((void **)&d_arr, n * sizeof(int));
    cudaMemcpy(d_arr, h_arr, n * sizeof(int), cudaMemcpyHostToDevice);

    int threads = 256;
    int pairs = (n + 1) / 2;
    int blocks = (pairs + threads - 1) / threads;

    for (int phase = 0; phase < n; phase++) {
        oddEvenStep<<<blocks, threads>>>(d_arr, n, phase);
        cudaDeviceSynchronize();
    }

    cudaMemcpy(h_arr, d_arr, n * sizeof(int), cudaMemcpyDeviceToHost);

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

    cudaFree(d_arr);
    free(h_arr);
    return 0;
}
```

Output:

```
6CSEC1@debian:~/Documents/230905152_PPL/L5$ nvcc -o abc a3.cu
6CSEC1@debian:~/Documents/230905152_PPL/L5$ ./abc
Enter N: 6
Enter arr[0]: 64
Enter arr[1]: 34
Enter arr[2]: 25
Enter arr[3]: 12
Enter arr[4]: 22
Enter arr[5]: 11
11 12 22 25 34 64
```