CS3811 - High Performance Computing and Big Data Lab

Lab 8

> Name: M K Lokesh Kumar

> Registration No.: 2201113026

> Class: Cyber Security(Semester 5)

Experiment 1

Objective

To use Taylor's series to approximate the value of Sin functions for different inputs, implemented to be run on both CPU and GPU using CUDA.

Code

Written in C.

- CPU Implementation

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>

long long factorialCalc(int n) {
    if (n == 0) {
        return 1;
    }
    long long res = 1;
    for (int i = 1; i <= n; i++) {
        res *= i;
    }
    return res;
}

float sinApprox(float x, int p) {
    float res = 0.0;
    for (int i = 0; i < p; i++) {
        int exp = 2 * i + 1;
        float term = powf(-1, i) * powf(x, exp) / factorialCalc(exp);
        res += term;
    }
```

```
    return res;
}

void findSin(float* inp, float* res, int N, int p) {
    for (int i = 0; i < N; i++) {
        res[i] = sinApprox(inp[i], p);
    }
}

void testTiming(int N, int p) {
    float *x, *res;

    x = (float *)malloc(N * sizeof(float));
    res = (float *)malloc(N * sizeof(float));

    for (int i = 0; i < N; i++) {
        x[i] = (float)i / N;
    }

    clock_t start = clock();
    findSin(x, res, N, p);
    clock_t end = clock();
    double cpu_time = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("%d\t%d\t%f\n", N, p, cpu_time);

    free(x);
    free(res);
}

int main() {
    int min_p = 3, max_p = 100;
    int min_N = 1 << 2;
    int max_N = 1 << 15;

    printf("N\tp\tTime\n");

    for (int p = min_p; p <= max_p; p += 10) {
        for (int N = min_N; N <= max_N; N *= 2) {
            testTiming(N, p);
        }
    }

    return 0;
}
```

- GPU Implementation

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

```
__device__ long long factCalc_gpu(int n) {
    if (n == 0) return 1;
    long long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}

__device__ float sinApprox_gpu(float x, int p) {
    float res = 0.0;
    for (int i = 0; i < p; i++) {
        int exp = 2 * i + 1;
        float term = powf(-1, i) * powf(x, exp) / factCalc_gpu(exp);
        res += term;
    }
    return res;
}

__global__ void gpu_findSin(float* arr, float* res, int N, int p) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < N) {
        res[i] = sinApprox_gpu(arr[i], p);
    }
}

void testTiming(int N, int p) {

    float *arr, *result, *d_arr, *d_result;
    size_t size = N * sizeof(float);

    arr = (float *)malloc(size);
    result = (float *)malloc(size);

    for (int i = 0; i < N; i++) {
        arr[i] = (float)i / N;
    }

    cudaMalloc((void **)&d_arr, size);
    cudaMalloc((void **)&d_result, size);

    cudaMemcpy(d_arr, arr, size, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start);

    gpu_findSin<<<blocksPerGrid, threadsPerBlock>>>(d_arr, d_result, N, p);

    cudaEventRecord(stop);
```

```
    cudaEventSynchronize(stop);

    cudaMemcpy(result, d_result, size, cudaMemcpyDeviceToHost);

    float time = 0;
    cudaEventElapsedTime(&time, start, stop);
    time /= 1000;

    printf("%d\t%d\t%f\n", N, p, time);

    cudaFree(d_arr);
    cudaFree(d_result);
    free(arr);
    free(result);
}

int main() {
    int min_p = 3, max_p = 100;
    int min_N = 1 << 2;
    int max_N = 1 << 15;

    printf("N\tp\tTime\n");
    for (int p = min_p; p <= max_p; p += 10) {
        for (int N = min_N; N <= max_N; N *= 2) {
            testTiming(N, p);
        }
    }

    return 0;
}
```

Output

- CPU Implementation

N	p	Time
4	3	0.000019
8	3	0.000002
16	3	0.000004
32	3	0.000005
64	3	0.000011
128	3	0.000020
256	3	0.000039
512	3	0.000077
1024	3	0.000153
2048	3	0.000307
4096	3	0.000615
8192	3	0.001236
16384	3	0.002563
32768	3	0.005779
4	13	0.000008
8	13	0.000011
16	13	0.000020
32	13	0.000040
64	13	0.000093
128	13	0.000157
256	13	0.000320
512	13	0.000639
1024	13	0.001259
2048	13	0.001671
4096	13	0.002732
8192	13	0.005286
16384	13	0.009945
32768	13	0.020284
4	23	0.000007
8	23	0.000014
16	23	0.000027
32	23	0.000053
64	23	0.000114
128	23	0.000209
256	23	0.000429
512	23	0.000842
1024	23	0.001690
2048	23	0.003358
4096	23	0.006754
8192	23	0.013455
16384	23	0.031302
32768	23	0.059614
4	33	0.000014
8	33	0.000026
16	33	0.000054
32	33	0.000101
64	33	0.000246
128	33	0.000405
256	33	0.000830
512	33	0.001714
1024	33	0.003816
2048	33	0.007652
4096	33	0.014413

```
8192     33       0.030969
16384    33       0.052328
32768    33       0.124311
4        43       0.000021
8        43       0.000041
16       43       0.000082
32       43       0.000163
64       43       0.000324
128      43       0.000650
256      43       0.001292
512      43       0.002645
1024     43       0.005239
2048     43       0.010386
4096     43       0.020950
8192     43       0.048430
16384    43       0.083745
32768    43       0.178270
4        53       0.000039
8        53       0.000078
16       53       0.000154
32       53       0.000320
64       53       0.000626
128      53       0.001265
256      53       0.002594
512      53       0.005308
1024     53       0.011106
2048     53       0.020751
4096     53       0.037574
8192     53       0.061935
16384    53       0.125284
32768    53       0.278472
4        63       0.000040
8        63       0.000084
16       63       0.000163
32       63       0.000423
64       63       0.000743
128      63       0.001413
256      63       0.002669
512      63       0.005314
1024     63       0.010875
2048     63       0.021446
4096     63       0.041859
8192     63       0.093168
16384    63       0.197453
32768    63       0.350801
4        73       0.000053
8        73       0.000108
16       73       0.000216
32       73       0.000424
64       73       0.000851
128      73       0.001704
256      73       0.003539
512      73       0.006986
1024     73       0.014166
```

2048	73	0.028289
4096	73	0.055141
8192	73	0.130708
16384	73	0.229080
32768	73	0.469579
4	83	0.000069
8	83	0.000139
16	83	0.000276
32	83	0.000579
64	83	0.001108
128	83	0.002205
256	83	0.004424
512	83	0.008769
1024	83	0.017956
2048	83	0.049358
4096	83	0.092150
8192	83	0.139402
16384	83	0.302841
32768	83	0.601944
4	93	0.000085
8	93	0.000174
16	93	0.000342
32	93	0.000689
64	93	0.001403
128	93	0.002746
256	93	0.005511
512	93	0.011425
1024	93	0.029769
2048	93	0.050363
4096	93	0.087259
8192	93	0.198122
16384	93	0.361463
32768	93	0.762004

- GPU Implementation

```
32768	63	0.000779
4	73	0.000236
8	73	0.000241
16	73	0.000244
32	73	0.000247
64	73	0.000248
128	73	0.000242
256	73	0.000305
512	73	0.000311
1024	73	0.000315
2048	73	0.000301
4096	73	0.000305
8192	73	0.000302
16384	73	0.000527
32768	73	0.001010
4	83	0.000284
8	83	0.000290
16	83	0.000293
32	83	0.000296
64	83	0.000295
128	83	0.000287
256	83	0.000379
512	83	0.000379
1024	83	0.000387
2048	83	0.000377
4096	83	0.000372
8192	83	0.000370
16384	83	0.000671
32768	83	0.001294
4	93	0.000336
8	93	0.000342
16	93	0.000347
32	93	0.000349
64	93	0.000347
128	93	0.000338
256	93	0.000471
512	93	0.000465
1024	93	0.000463
2048	93	0.000462
4096	93	0.000466
8192	93	0.000458
16384	93	0.000827
32768	93	0.001600
```

```
256 33  0.000099
512 33  0.000097
1024    33  0.000097
2048    33  0.000099
4096    33  0.000095
8192    33  0.000098
16384   33  0.000136
32768   33  0.000246
4   43  0.000117
8   43  0.000121
16  43  0.000125
32  43  0.000129
64  43  0.000130
128 43  0.000126
256 43  0.000138
512 43  0.000141
1024    43  0.000134
2048    43  0.000140
4096    43  0.000137
8192    43  0.000139
16384   43  0.000207
32768   43  0.000392
4   53  0.000152
8   53  0.000156
16  53  0.000161
32  53  0.000164
64  53  0.000164
128 53  0.000161
256 53  0.000185
512 53  0.000184
1024    53  0.000188
2048    53  0.000188
4096    53  0.000183
8192    53  0.000187
16384   53  0.000300
32768   53  0.000572
4   63  0.000192
8   63  0.000196
16  63  0.000200
32  63  0.000205
64  63  0.000205
128 63  0.000197
256 63  0.000247
512 63  0.000244
1024    63  0.000242
2048    63  0.000244
4096    63  0.000234
8192    63  0.000246
16384   63  0.000408
```

N	p	Time
4	3	0.000234
8	3	0.000016
16	3	0.000013
32	3	0.000012
64	3	0.000012
128	3	0.000012
256	3	0.000013
512	3	0.000012
1024	3	0.000012
2048	3	0.000013
4096	3	0.000012
8192	3	0.000013
16384	3	0.000010
32768	3	0.000017
4	13	0.000032
8	13	0.000031
16	13	0.000031
32	13	0.000031
64	13	0.000033
128	13	0.000034
256	13	0.000037
512	13	0.000037
1024	13	0.000037
2048	13	0.000036
4096	13	0.000036
8192	13	0.000033
16384	13	0.000035
32768	13	0.000060
4	23	0.000055
8	23	0.000057
16	23	0.000060
32	23	0.000062
64	23	0.000063
128	23	0.000065
256	23	0.000068
512	23	0.000065
1024	23	0.000063
2048	23	0.000063
4096	23	0.000068
8192	23	0.000067
16384	23	0.000077
32768	23	0.000142
4	33	0.000089
8	33	0.000094
16	33	0.000096
32	33	0.000097
64	33	0.000100
128	33	0.000097