



Bouns Project

CSE 336 Distributed Computing

Name: **Eslam Ahmed Genedy** ID: **17p6043**

Experiment Title: OpenCL and PI

Date: 18 / 5 /2018

Introduction :

The project is built using Open CL 1.2 as a framework.

Using Intel ® HD Graphics 630 for computing

Split the equation into a 1D array

for the final validation, get the summation of the array multiply it by 4/N

Comparing the output with the defined PI

All values were squared \Rightarrow is to validate the HelloWorld example

The error is equal to 0.001991

repo:[Open-GI and PI](#)

```
Intel Inc. : Intel(R) HD Graphics 630
All values were squared.
myPI: 3.139602, PI 3.141593
Program ended with exit code: 0
```

```

#include <math.h>

#include <OpenCL/OpenCL.h>

#include "helloWorld.cl.h"
#include "calPI.cl.h"

#define PI 3.1415926535897932384
#define N 1024
#define EPS 1e-6

static int validate_square(int i, float* input, cl_float* output)
{
    int i;
    for (i = 0; i < N; i++) {
        if (fabs(output[i] - input[i] * input[i]) > EPS) {
            fprintf(stderr, "Error: Element %d did not match expected output.\n", i);
            fprintf(stderr, "Was: %.4f, expected %.4f\n", output[i], input[i]*input[i]);
            fflush(stderr);
            return 0;
        }
    }
    return 1;
}

static int validate_pi(cl_float* in1, cl_float* out)
{
    int i;
    float myPI = 0.0;
    for (i = 0; i < N; i++) {
        myPI += output[i];
    }
    myPI = myPI * (4.0/1024.0);

    fprintf(stderr, "myPI: %f, PI %f\n", myPI, PI);
    fflush(stderr);
    return 0;
}

```

```

Intel(R) Znu. : Intel(R) HD Graphics 630
All values were squared.
myPI: 3.139682, PI 3.141593
Program ended with exit code: 0

```

Code of the kernel (calPI.cl.h) :

1. $\text{root} (1 - \text{square}(i/n))$
2. the value get added to the output
3. kernel void calPI(global float * in1, global float * in2, global float * out)
4. {
5. size_t i = get_global_id(0);
6. $\text{out}[i] = \text{sqrt}(1 - \text{pow}((\text{in1}[i]/\text{in2}[i]), 2));$
7. }

Code of the host (main.c) :

1. //

```

2. // main.c
3. // HelloWorldCL
4. // Copyright (c) 2019 Daboor All rights reserved.
5. //
6.
7. #include <stdio.h>
8. #include <math.h>
9.
10. #include <OpenCL/OpenCL.h>
11.
12. #include "helloworld.cl.h"
13. #include "calPI.cl.h"
14.
15.
16. #define PI 3.14159265358979323846
17. #define N 1024
18. #define EPS 1e-3
19.
20.
21. static int validate_square(cl_float* input, cl_float* output)
22. {
23.     int i;
24.     for (i = 0; i < N; i++) {
25.         if (fabs(output[i] - input[i] * input[i]) > EPS) {
26.             fprintf(stderr, "Error: Element %d did not match expected output.\n", i);
27.             fprintf(stderr, "Saw: %1.4f, expected %1.4f\n", output[i], input[i]*input[i]);
28.             fflush(stderr);
29.             return 0;
30.         }
31.     }
32.     return 1;
33. }
34.
35. static int validate_pi(cl_float* in1, cl_float* output)
36. {
37.     int i;
38.     float myPI = 0.0 ;
39.     for (i = 1; i < N+1; i++) {
40.         myPI += output[i] ;
41.     }
42.     myPI= myPI * (4.0/1024.0);
43.
44.     fprintf(stderr, "myPI: %f, PI %f \n", myPI , PI);
45.     fflush(stderr);

```

```

46.     return 0;
47.
48.
49. }
50.
51.
52.
53. static void print_device_info(cl_device_id device)
54. {
55.     char name[128];
56.     char vendor[128];
57.
58.     clGetDeviceInfo(device, CL_DEVICE_NAME, 128, name, NULL);
59.     clGetDeviceInfo(device, CL_DEVICE_VENDOR, 128, vendor, NULL);
60.
61.     fprintf(stdout, "%s : %s\n", vendor, name);
62. }
63.
64. #pragma mark -
65. #pragma mark Hello World - Sample 1
66.
67. int main(int argc, const char** argv)
68. {
69.     int i;
70.     const size_t byte_size = sizeof(cl_float) * N;
71.
72.     dispatch_queue_t queue = gcl_create_dispatch_queue(CL_DEVICE_TYPE_GPU, NULL);
73.     if (queue == NULL)
74.         queue = gcl_create_dispatch_queue(CL_DEVICE_TYPE_CPU, NULL);
75.
76.     cl_device_id gpu = gcl_get_device_id_with_dispatch_queue(queue);
77.     print_device_info(gpu);
78.
79.     // ===== HelloWorld =====
80.     float *test_in = (float*)malloc(sizeof(cl_float) * N);
81.     for (i = 0; i < N; i++)
82.         test_in[i] = (cl_float)i;
83.
84.     float *test_out = (float*)malloc(sizeof(cl_float) * N);
85.
86.     void *mem_in = gcl_malloc(sizeof(cl_float) * N, test_in, CL_MEM_READ_ONLY |
        CL_MEM_COPY_HOST_PTR);
87.     void *mem_out = gcl_malloc(sizeof(cl_float) * N, NULL, CL_MEM_WRITE_ONLY);
88.

```

```

89.     dispatch_sync(queue, ^{
90.         size_t wgs;
91.         gcl_get_kernel_block_workgroup_info(square_kernel, CL_KERNEL_WORK_GROUP_SIZE,
           sizeof(wgs), &wgs, NULL);
92.         cl_ndrange range = { 1                                //number of dimensions
93.                               , {0, 0, 0}                      //offsets in dimensions
94.                               , {N, 0, 0}                      //global range
95.                               , {wgs, 0, 0}};                  //local size of workgroup
96.         square_kernel(&range, (cl_float*)mem_in, (cl_float*)mem_out);
97.         gcl_memcpy(test_out, mem_out, sizeof(cl_float) * N);
98.     });
99.
100.    if (validate_square(test_in, test_out))
101.        fprintf(stdout, "All values were squared.\n");
102.
103.    gcl_free(mem_in);
104.    gcl_free(mem_out);
105.
106.    free(test_in);
107.    free(test_out);
108.
109.                                     //      ===== PI
           =====
110.    float *test_in1 = (float*)malloc(byte_size);
111.    float *test_in2 = (float*)malloc(byte_size);
112.
113.    for (i = 1; i < N+1; i++) {
114.        test_in1[i] = (cl_float)i;           // i value from range 1 to 1024 as
115.        test_in2[i] = (cl_float)N ;          // Passing the value of N to each core
116.    }
117.
118.    test_out = (float*)malloc(byte_size);
119.
120.    void *mem_in1 = gcl_malloc(byte_size, test_in1, CL_MEM_READ_ONLY |
CL_MEM_COPY_HOST_PTR);
121.    void *mem_in2 = gcl_malloc(byte_size, test_in2, CL_MEM_READ_ONLY |
CL_MEM_COPY_HOST_PTR);
122.    mem_out = gcl_malloc(byte_size, NULL, CL_MEM_WRITE_ONLY );      //write only for
the output
123.
124.    dispatch_sync(queue, ^{
125.        size_t wgs;
126.        gcl_get_kernel_block_workgroup_info(calPI_kernel, CL_KERNEL_WORK_GROUP_SIZE,
           sizeof(wgs), &wgs, NULL);

```

```

127.         cl_ndrange range = { 1                                //number of dimensions
128.                                     , {0, 0, 0}                  //offsets in dimensions
129.                                     , {N, 0, 0}                  //global range
130.                                     , {wgs, 0, 0}};              //local size of workgroup
131.         calPI_kernel(&range,    (cl_float*)mem_in1,    (cl_float*)mem_in2,
        (cl_float*)mem_out);
132.         gcl_memcpy(test_out, mem_out, byte_size);
133.     });
134.
135.     if (validate_pi(test_in1, test_out))
136.         fprintf(stdout, "All values were summed.\n");
137.
138.         //      =====free      the
        meomry=====
139.         gcl_free(mem_in1);
140.         gcl_free(mem_in2);
141.         gcl_free(mem_out);
142.
143.         free(test_in1);
144.         free(test_in2);
145.         free(test_out);
146.
147.
148.
149.     return 0;
150. }
151.
152.

```