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

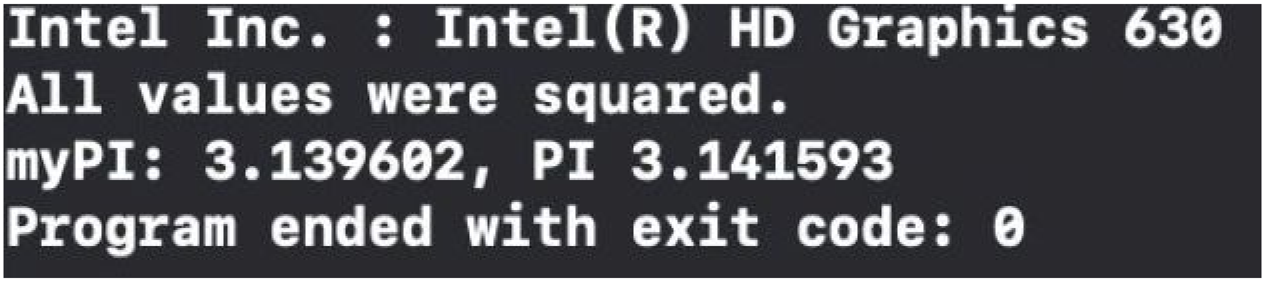
for the final validation, get the summation of the array multiply it by 4/N

Comparing the output with the defined PI

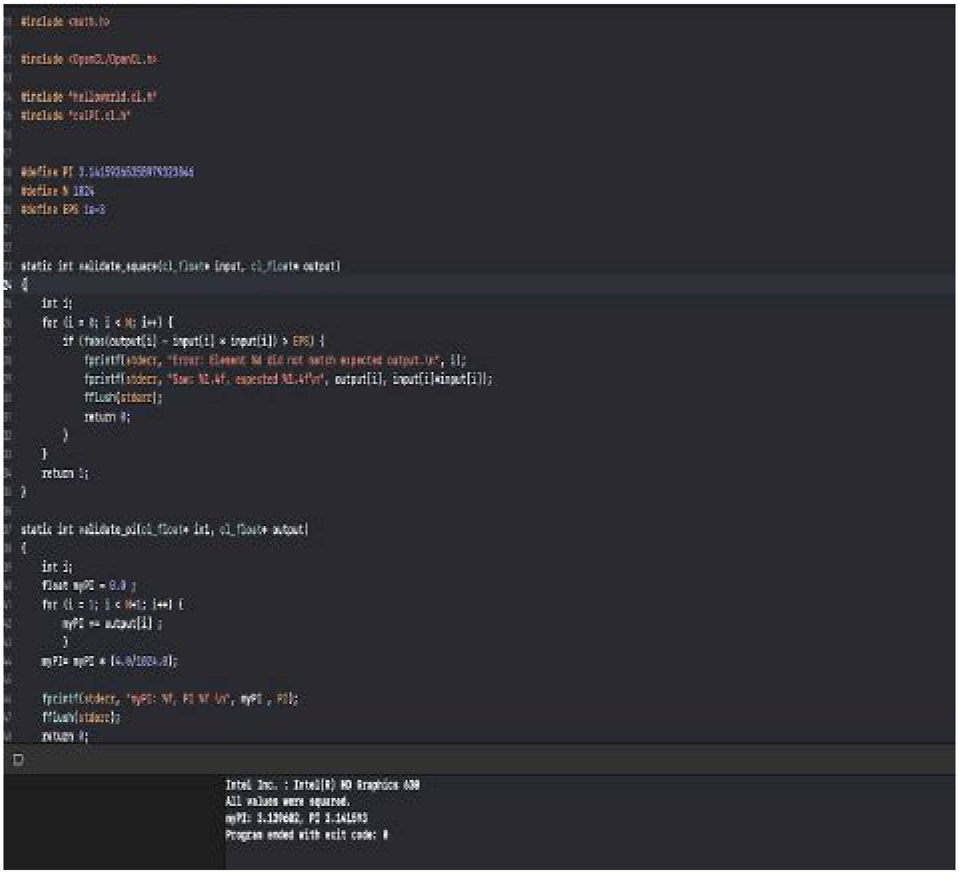
All values were squared ⇒ is to valiedate the HelloWorld example

The error is equal to 0.001991

repo:Open-Gl 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 3.14159265358979323846
#define N 1024
#define EPS 1e-3


static int validate_square(cl_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, "Saw %1.4f, expected %1.4f\n", output[i], input[i]*input[i]);
            fflush(stderr);
            return 0;
        }
    }
    return 1;
}

static int validate_pi(cl_float* in1, cl_float* output)
{
    int i;
    float myPI = 0.0 ;
    for (i = 1; i < N+1; i++) {
        myPI += output[i] ;
        }
    myPI= myPI * (4.0/1024.0);

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

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

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

```
root ( 1- square(i/n) )

the value get added to the output

kernel void calPI( global float * in1, global float * in2, global float * out)

{

size_t i = get_global_id(0);

out[i]=sqrt(1-pow((in1[i]/in2[i]),2));

}
```

Code of the host (main.c) :

```
//
```

```c
//  main.c
//  HelloWorldCL
//  Copyright (c) 2019 Daboor All rights reserved.
//

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

#include <OpenCL/OpenCL.h>

#include "helloworld.cl.h"
#include "calPI.cl.h"


#define PI 3.14159265358979323846
#define N 1024
#define EPS 1e-3


static int validate_square(cl_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, "Saw: %1.4f, expected %1.4f\n", output[i], input[i]*input[i]);
            fflush(stderr);
            return 0;
        }
    }
    return 1;
}

static int validate_pi(cl_float* in1, cl_float* output)
{
    int i;
    float myPI = 0.0 ;
    for (i = 1; i < N+1; i++) {
        myPI += output[i] ;
        }
    myPI= myPI * (4.0/1024.0);

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


}



static void print_device_info(cl_device_id device)
{
    char name[128];
    char vendor[128];

    clGetDeviceInfo(device, CL_DEVICE_NAME, 128, name, NULL);
    clGetDeviceInfo(device, CL_DEVICE_VENDOR, 128, vendor, NULL);

    fprintf(stdout, "%s : %s\n", vendor, name);
}

#pragma mark -
#pragma mark Hello World - Sample 1

int main(int argc, const char** argv)
{
    int i;
    const size_t byte_size = sizeof(cl_float) * N;

    dispatch_queue_t queue = gcl_create_dispatch_queue(CL_DEVICE_TYPE_GPU, NULL);
    if (queue == NULL)
        queue = gcl_create_dispatch_queue(CL_DEVICE_TYPE_CPU, NULL);

    cl_device_id gpu = gcl_get_device_id_with_dispatch_queue(queue);
    print_device_info(gpu);

    // ======== HelloWorld =========
    float *test_in = (float*)malloc(sizeof(cl_float) * N);
    for (i = 0; i < N; i++)
        test_in[i] = (cl_float)i;

    float *test_out = (float*)malloc(sizeof(cl_float) * N);

    void *mem_in = gcl_malloc(sizeof(cl_float) * N, test_in, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR);
    void *mem_out = gcl_malloc(sizeof(cl_float) * N, NULL, CL_MEM_WRITE_ONLY);

    dispatch_sync(queue, ^{
        size_t wgs;
        gcl_get_kernel_block_workgroup_info(square_kernel, CL_KERNEL_WORK_GROUP_SIZE, sizeof(wgs), &wgs, NULL);
        cl_ndrange range = { 1                               //number of dimensions
                           , {0, 0, 0}                       //offsets in dimensions
                           , {N, 0, 0}                       //global range
                           , {wgs, 0, 0}};                   //local size of workgroup
        square_kernel(&range, (cl_float*)mem_in, (cl_float*)mem_out);
        gcl_memcpy(test_out, mem_out, sizeof(cl_float) * N);
    });

    if (validate_square(test_in, test_out))
        fprintf(stdout, "All values were squared.\n");

    gcl_free(mem_in);
    gcl_free(mem_out);

    free(test_in);
    free(test_out);

    // ========================================= PI ==========================================
    float *test_in1 = (float*)malloc(byte_size);
    float *test_in2 = (float*)malloc(byte_size);

    for (i = 1; i < N+1; i++) {
        test_in1[i] = (cl_float)i;            // i value from range 1 to 1024 as
        test_in2[i] = (cl_float)N ;           // Passing the value of N to each core
    }

    test_out = (float*)malloc(byte_size);

    void *mem_in1 = gcl_malloc(byte_size, test_in1, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR);
    void *mem_in2 = gcl_malloc(byte_size, test_in2, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR);
    mem_out = gcl_malloc(byte_size, NULL, CL_MEM_WRITE_ONLY );       //write only for the output

    dispatch_sync(queue, ^{
        size_t wgs;
        gcl_get_kernel_block_workgroup_info(calPI_kernel, CL_KERNEL_WORK_GROUP_SIZE, sizeof(wgs), &wgs, NULL);
        cl_ndrange range = { 1                              //number of dimensions
                           , {0, 0, 0}              //offsets in dimensions
                           , {N, 0, 0}              //global range
                           , {wgs, 0, 0}};          //local size of workgroup
        calPI_kernel(&range, (cl_float*)mem_in1, (cl_float*)mem_in2, (cl_float*)mem_out);
        gcl_memcpy(test_out, mem_out, byte_size);
    });

    if (validate_pi(test_in1, test_out))
        fprintf(stdout, "All values were summed.\n");

    // ====================================free the meomry===========================================
    gcl_free(mem_in1);
    gcl_free(mem_in2);
    gcl_free(mem_out);

    free(test_in1);
    free(test_in2);
    free(test_out);



    return 0;
}

```