Grupa ćwicz. I	*Nr. sprawozdania.* V	*Kierunek i rok* Informatyka Stosowana stopień II / semestr III	*Data wykonania.* 7/5/2015	*Data odbioru*
Imię i nazwiska. Dawid Wacławik			*Ocena i uwagi*	

Opanowanie umiejętności programowania kart graficznych w środowisku OpenCL.

Plik Makefile

```
# optimization and other system dependent options
#include  ../platform_files/make.$(OWW_ARCH)

NAME = Hello_GPU
LIB = -L/opt/cuda7/lib64 -lOpenCL
INC = -I/opt/cuda7/include/ -I./include/

program: obj/main.o obj/opencl_init.o include/opencl_local.h
    $(CC) $(LDFL) obj/main.o obj/opencl_init.o $(LIB) -o $(NAME)

obj/main.o: main/main.c include/opencl_local.h
    $(CC) $(CFL) -c main/main.c  $(INC) -o obj/main.o

obj/opencl_init.o: opencl_init/opencl_init.c include/opencl_local.h
    $(CC) $(CFL) -c opencl_init/opencl_init.c  $(INC) -o obj/opencl_init.o

clean:
    rm -f obj/*
    rm -f $(NAME)
```

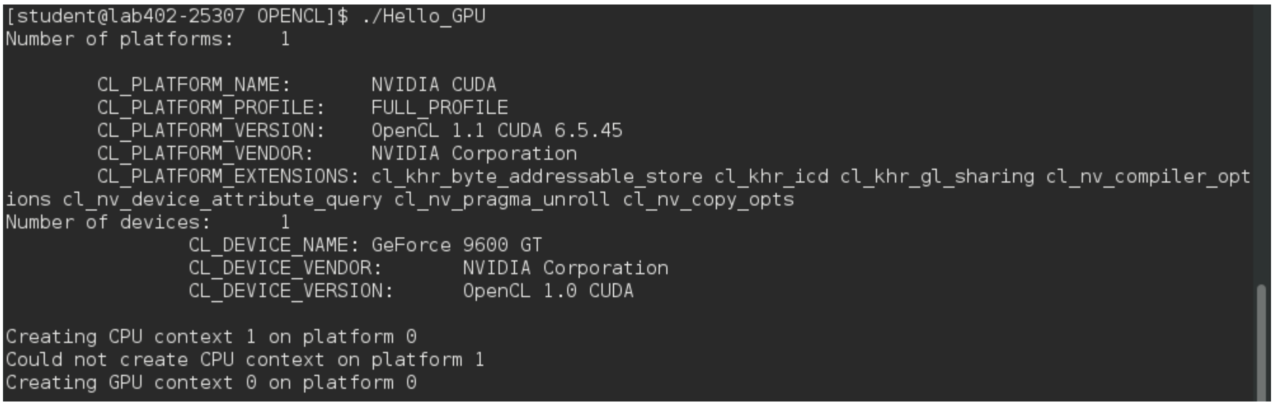
Plik main.c

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

#include <CL/cl.h>

// main program controlling execution of CPU code and OpenCL kernels
int main(int argc, char** argv)
{
  cl_uint number_of_contexts = 2;
  cl_context context = NULL;
  cl_context list_of_contexts[2] = {0,0};
  cl_command_queue commandQueue = 0;
  cl_program program = 0;
  cl_uint number_of_devices;
  cl_device_id device = 0;
  cl_device_id *list_of_devices;
  cl_device_type type;
  cl_kernel kernel = 0;
  cl_mem memObjects[3] = { 0, 0, 0 };
  cl_int retval;
  int icon, idev;
  cl_uint numPlatforms;
  cl_platform_id * platformIds;
  cl_uint i,j;

  // Create OpenCL contexts
  int Monitor = 1;

  // First, query the total number of platforms
  retval = clGetPlatformIDs(0, (cl_platform_id *) NULL, &numPlatforms);

  // Next, allocate memory for the installed plaforms, and qeury
  // to get the list.
  platformIds = (cl_platform_id *)malloc(sizeof(cl_platform_id) * numPlatforms);

  // Then, query the platform IDs
  retval = clGetPlatformIDs(numPlatforms, platformIds, NULL);

  if(Monitor>=0){
    printf("Number of platforms: \t%d\n", numPlatforms);
  }

  // Iterate through the list of platforms displaying associated information
  for (i = 0; i < numPlatforms; i++) {

    if(Monitor>0){

      printf("\n");
      // First we display information associated with the platform
      DisplayPlatformInfo(
                                platformIds[i],
                                CL_PLATFORM_NAME,
                                "CL_PLATFORM_NAME");
      DisplayPlatformInfo(
                                platformIds[i],
                                CL_PLATFORM_PROFILE,
                                "CL_PLATFORM_PROFILE");
      DisplayPlatformInfo(
                                platformIds[i],
                                CL_PLATFORM_VERSION,
                                "CL_PLATFORM_VERSION");
      DisplayPlatformInfo(
                                platformIds[i],
                                CL_PLATFORM_VENDOR,
                                "CL_PLATFORM_VENDOR");
      DisplayPlatformInfo(
                                platformIds[i],
                                CL_PLATFORM_EXTENSIONS,
                                "CL_PLATFORM_EXTENSIONS" );

    }
  }

  // For the first platform
  int iplat = 0;

  // Query the set of devices associated with the platform
  retval = clGetDeviceIDs(
                                          platformIds[iplat],
                                          CL_DEVICE_TYPE_ALL,
                                          0,
                                          NULL,
                                          &number_of_devices);


  list_of_devices =
    (cl_device_id *) malloc (sizeof(cl_device_id) * number_of_devices);

  retval = clGetDeviceIDs(
                                          platformIds[iplat],
                                          CL_DEVICE_TYPE_ALL,
                                          number_of_devices,
                                          list_of_devices,
                                          NULL);

  if(Monitor>=0){
    printf("Number of devices: \t%d\n", number_of_devices);
  }

  // Iterate through each device, displaying associated information
  for (j = 0; j < number_of_devices; j++) {

    clGetDeviceInfo(list_of_devices[j], CL_DEVICE_TYPE,
                                sizeof(cl_device_type), &type, NULL);

    if(Monitor>0){

      DisplayDeviceInfo(
                                list_of_devices[j],
                                CL_DEVICE_NAME,
                                "CL_DEVICE_NAME");

      DisplayDeviceInfo(
                                list_of_devices[j],
                                CL_DEVICE_VENDOR,
                                "CL_DEVICE_VENDOR");

      DisplayDeviceInfo(
                                list_of_devices[j],
                                CL_DEVICE_VERSION,
                                "CL_DEVICE_VERSION");

      printf("\n");
    }
  }

  // Next, create OpenCL contexts on platforms
  cl_context_properties contextProperties[] = {
    CL_CONTEXT_PLATFORM,
    (cl_context_properties)platformIds[iplat],
    0
  };

  if(Monitor>0){
    printf("Creating CPU context %d on platform %d\n", 1, iplat);
  }

  list_of_contexts[1] =
    clCreateContextFromType(contextProperties,
                                            CL_DEVICE_TYPE_CPU, NULL, NULL, &retval);

  if(Monitor>=0 && retval != CL_SUCCESS){
    printf("Could not create CPU context on platform %d\n", i);
  }

  if(Monitor>0){
    printf("Creating GPU context 0 on platform %d\n", iplat);
  }
```

```
list_of_contexts[0] =
  clCreateContextFromType(contextProperties,
                                                    CL_DEVICE_TYPE_GPU, NULL, NULL, &retval);

if(Monitor>=0 && retval != CL_SUCCESS){
  printf("Could not create GPU context on platform %d\n", i);
}

// in a loop over devices of the seleceted platform
for(idev=0; idev<number_of_devices;idev++){

  if(Monitor>0){
    printf("\nFor context %d and device %d:\n",
               idev, idev);
  }
  device = list_of_devices[idev];
  icon = idev;

  // choose OpenCL context on first available platform
  context = list_of_contexts[icon];

  if(context !=0){

    commandQueue = clCreateCommandQueue(context, device, 0, NULL);
    if (commandQueue == NULL) {
          printf("Failed to create commandQueue for device %d\n", idev);
          exit(0);
    }

    if(Monitor>0){
          printf("Reading program from source\n");
    }

    // read source code from file
    FILE *fp;
    char* source;
    long int size;

    fp = fopen("HelloWorld.cl", "rb");
    if(!fp) {
          printf("Could not open kernel file\n");
          exit(-1);
    }
    int status = fseek(fp, 0, SEEK_END);
    if(status != 0) {
          printf("Error seeking to end of file\n");
          exit(-1);
    }
    size = ftell(fp);
    if(size < 0) {
          printf("Error getting file position\n");
          exit(-1);
    }

    rewind(fp);

    source = (char *)malloc(size + 1);

    int i;
    for (i = 0; i < size+1; i++) {
          source[i]='\0';
    }

    if(source == NULL) {
          printf("Error allocating space for the kernel source\n");
          exit(-1);
    }

    fread(source, 1, size, fp);
    source[size] = '\0';

    cl_program program = clCreateProgramWithSource(context, 1,
                                                                                       &source,
                                                                                       NULL, NULL);
    if (program == NULL)
          {
            printf("Failed to create CL program from source.\n");
            exit(-1);
```

```
     }

if(Monitor>0){
     printf("Creating program and kernel\n");
}
// build program (passing options to compiler if necessary
retval = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
char* buildLog; size_t size_of_buildLog;
clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG,
                         0, NULL, &size_of_buildLog);
buildLog = malloc(size_of_buildLog+1);
clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG,
                         size_of_buildLog, buildLog, NULL);
buildLog[size_of_buildLog]= '\0';
printf("Kernel buildLog: %s\n", buildLog);
if (retval != CL_SUCCESS)
     {
      printf("Error in kernel\n");
      clReleaseProgram(program);
      exit(-1);
     }


// Create OpenCL kernel
kernel = clCreateKernel(program, "hello_kernel", NULL);
if (kernel == NULL)
     {
      printf("Failed to create kernel.\n");
      exit(0);
     }

if(Monitor>0){
     printf("Creating memory objects\n");
}
// Create memory objects that will be used as arguments to
// kernel.  First create host memory arrays that will be
// used to store the arguments to the kernel
float result[1];
float a[1];
float b[1];
a[0] = 2;
b[0] = 2;

memObjects[0] = clCreateBuffer(context, CL_MEM_READ_ONLY ,
                                  sizeof(float), NULL, NULL);
memObjects[1] = clCreateBuffer(context, CL_MEM_READ_ONLY ,
                                  sizeof(float), NULL, NULL);
memObjects[2] = clCreateBuffer(context, CL_MEM_READ_WRITE,
                                  sizeof(float), NULL, NULL);

if (memObjects[0]==NULL || memObjects[1]==NULL || memObjects[2]==NULL){
     printf("Error creating memory objects.\n");
     return 0;
}

if(Monitor>0){
     printf("Sending kernel arguments\n");
}
retval = clEnqueueWriteBuffer(
                                  commandQueue,
                                  memObjects[0],
                                  CL_FALSE,
                                  0,
                                  sizeof(float),
                                  a,
                                  0,
                                  NULL,
                                  NULL);

// Use clEnqueueWriteBuffer() to write input array B to
// the device buffer bufferB
retval = clEnqueueWriteBuffer(
                                  commandQueue,
                                  memObjects[1],
                                  CL_FALSE,
                                  0,
                                  sizeof(float),
                                  b,
                                                        0,
                                                        NULL,
                                                        NULL);

    // Set the kernel arguments (result, a, b)
    retval = clSetKernelArg(kernel, 0, sizeof(cl_mem), &memObjects[0]);
    retval |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &memObjects[1]);
    retval |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &memObjects[2]);
    if (retval != CL_SUCCESS)
          {
            printf("Failed to Set the kernel arguments.\n");
            //Cleanup(context, commandQueue, program, kernel, memObjects);
            return 1;
          }

    if(Monitor>0){
          printf("Running the kernel!\n");
    }
    size_t globalWorkSize[1] = { 1 };
    size_t localWorkSize[1] = { 1 };

    // Queue the kernel up for execution across the array
    retval = clEnqueueNDRangeKernel(commandQueue, kernel, 1, NULL,
                                                         globalWorkSize, localWorkSize,
                                                         0, NULL, NULL);
    if (retval != CL_SUCCESS)
          {
            printf("Failed to queue kernel for execution.\n");
            //Cleanup(context, commandQueue, program, kernel, memObjects);
            return 1;
          }

    if(Monitor>0){
          printf("Transfering back results\n");
    }
    // Read the output buffer back to the Host
    retval = clEnqueueReadBuffer(commandQueue, memObjects[2], CL_TRUE,
                                                       0, sizeof(float), result,
                                                       0, NULL, NULL);
    if (retval != CL_SUCCESS)
          {
            printf("Failed to read result buffer.\n");
            //Cleanup(context, commandQueue, program, kernel, memObjects);
            return 1;
          }

    // Verify the output
    if(result[0]==4)  {
      printf("Output is correct: %lf + %lf = %lf\n",
                 a[0], b[0], result[0]);
    } else {
      printf("Output is incorrect: %lf + %lf != %lf\n",
                 a[0], b[0], result[0]);
    }

    for (i = 0; i < 3; i++)
          {
            if (memObjects[i] != 0)
        clReleaseMemObject(memObjects[i]);
          }
    if (commandQueue != 0)
      clReleaseCommandQueue(commandQueue);

    if (kernel != 0)
      clReleaseKernel(kernel);

    if (program != 0)
      clReleaseProgram(program);

  }
 }

 free(list_of_devices);
 free(platformIds);

 return 0;
}
```

Screen



```
[student@lab402-25307 OPENCL]$ ./Hello_GPU
Number of platforms:    1

        CL_PLATFORM_NAME:       NVIDIA CUDA
        CL_PLATFORM_PROFILE:    FULL_PROFILE
        CL_PLATFORM_VERSION:    OpenCL 1.1 CUDA 6.5.45
        CL_PLATFORM_VENDOR:     NVIDIA Corporation
        CL_PLATFORM_EXTENSIONS: cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_opt
ions cl_nv_device_attribute_query cl_nv_pragma_unroll cl_nv_copy_opts
Number of devices:      1
                CL_DEVICE_NAME: GeForce 9600 GT
                CL_DEVICE_VENDOR:       NVIDIA Corporation
                CL_DEVICE_VERSION:      OpenCL 1.0 CUDA

Creating CPU context 1 on platform 0
Could not create CPU context on platform 1
Creating GPU context 0 on platform 0
```

Wnioski

Kod został przeanalizowany i zmodyfikowany na zajęciach. Pierwszą modyfikacją była w pliku „**Makefile**” , zostały przypisane biblioteki do zmiennych LIB i INC. Kolejną modyfikacją było dodanie do pliku „**main.c**” DisplayPlatformInfo(platformIds[i], CL_PLATFORM_EXTENSIONS, "CL_PLATFORM_EXTENSIONS"); dzięki czemu został wyświetlony dodatkowy parametr platformy.

Nie udało mi się na zajęciach zmodyfikować w odpowiedni sposób kernela, starałem się przerobić program w domu, lecz miałem problemy z zainstalowaniem „**cuda**”.

Z otrzymanych wyników możemy odczytać wersje OpenCL i Cuda, nazwe itp. Również dostajemy informacje o rodzaju zainstalowanej karty graficznej na komputerze.