Московский Авиационный Институт
(Государственный Технический Университет)

Факультет прикладной математики и физики.
Кафедра вычислительной математики и программирования.

Лабораторная работа №1

по курсу «Программирование графических процессоров»

Освоение фреймворка OpenCL для написания компьютерных программ, связанных с параллельными вычислениями .

VII семестр.

Студент Баскаков О.А.
Группа 08-406
Преподаватель Семенов С.А.

Москва, 2011.

Постановка задачи

В рамках данной лабораторной работы требуется ознакомиться со средствами написания программ на языке OpenCL, написать и отладить примитивную программу-пример, содержащую базовые принципы программирования графических процессоров с использованием параллельных вычислений.
К выполнению поставленной задачи предъявляются следующие требования:

- Программа должна выполняться параллельно
- Программа должна распределять память
- Программа должна использовать векторные операции

Описание

Для решения поставленной задачи программа:

- Определяет наличие необходимого устройства (видеокарты);
- Выделяет необходимое количество памяти;
- Определяет размерность блоков;
- Определяет число необходимых блоков;
- Запускает ядро, передавая ему число блоков и число нитей в каждом блоке;
- В рамках ядра производит необходимые вычисления параллельно;
- Копирует результаты работы программы из памяти видеокарты;
- Высвобождает память видеокарты;
- Выводит на экран результаты работы программы;

Ключевыми отличиями OpenCL от C99 являются:

- Отсутствие поддержки указателей на функции, рекурсии, битовых полей, массивов переменной длины (VLA), стандартных заголовочных файлов;
- Расширения языка для параллелизма: векторные типы, синхронизация, функции для Work-items/Work-Groups;
- Квалификаторы типов памяти: __global, __local, __constant, __private ;
- Иной набор встроенных функций ;

В реализации была использована обертка PyOpenCL над стандартными C++ библиотеками.

Вариант задания:

1. Сложение векторов
2. Сложение с векторизацией операций

Код программы на языке Python

0. Инициализация:

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import pyopencl as cl
import pyopencl.array as cl_array
import numpy as np
import numpy.linalg as la

platform = cl.get_platforms()[0]
devices = platform.get_devices(device_type=my_type)
ctx = cl.Context(devices = devices)
```

1. OpenCL __kernels:

```
code = """
__kernel void
sum(__global const float *a,
    __global const float *b,
    __global float *c)
{
  int gid = get_global_id(0);
  c[gid] = a[gid] + b[gid];
}

__kernel void
sum4(__global const float4 *a,
    __global const float4 *b,
    __global float4 *c)
{
  int gid = get_global_id(0);
  c[gid] = a[gid] + b[gid];
}
"""
```

2. Функция main:

```
def main():
      global prg, code
      # initialize_CL
      ctx = cl_init()
      cp = cl.command_queue_properties
      queue = cl.CommandQueue(ctx, properties=cp.PROFILING_ENABLE)
      prg = cl.Program(ctx, code).build()
      # generate long random vectors
      size = 100500*16
      a = np.random.rand(size).astype(np.float32)
      b = np.random.rand(size).astype(np.float32)
      a_plus_b = np.empty_like(a)
      # create buffers
      mf = cl.mem_flags
      a_buf = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=a)
      b_buf = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=b)
      dest_buf = cl.Buffer(ctx, mf.WRITE_ONLY, b.nbytes)

      print("...Calcuate 16*100500 elem's vector")
      exec_evt = prg.sum(queue, a.shape, (16,), a_buf, b_buf, dest_buf).wait()
      elapsed = 1e-9*(exec_evt.profile.end - exec_evt.profile.start)
      print("Execution time of test1: %g second" % elapsed)
      cl.enqueue_copy(queue, a_plus_b, dest_buf)
```

```
    print("...Start vectorized kernel")
    # call OpenCL kernel
    exec_evt=prg.sum4(queue,(a.shape[0]//4,),(16,),a_buf,b_buf,dest_buf)
    exec_evt .wait()
    elapsed = 1e-9*(exec_evt.profile.end - exec_evt.profile.start)
    print("Execution time of test2: %g second" % elapsed)

    cl.enqueue_copy(queue, a_plus_b, dest_buf)

    # compare result with native numpy
    print('error = ' + str(la.norm(a_plus_b - (a+b))))
```

Протокол

```
oleg@spetz:~/CL/lab1$ python lab1.py
================================================================
Platform name: AMD Accelerated Parallel Processing
Platform vendor: Advanced Micro Devices, Inc.
Platform version: OpenCL 1.1 AMD-APP-SDK-v2.5 (684.213)
----------------------------------------------------------------
Device name: Redwood
Device type: GPU
Local memory: 32 KB
Device memory: 512 MB
Device max clock speed:550 MHz
Device compute units:5
...Calcuate 16*100500 elem's vector
Execution time of test1: 0.00780422 second
...Start vectorized kernel
Execution time of test2: 0.00544356 second
error = 0.0
Testing:
[ 0.59591997  0.26145452  0.19505562]
+
[ 0.80712444  0.37728721  0.365228  ]
=
[ 1.40304446  0.63874173  0.5602836 ]
```

Выводы

Цель OpenCL состоит в том, чтобы дополнить OpenGL и OpenAL, которые являются открытыми отраслевыми стандартами для трёхмерной компьютерной графики и звука, пользуясь возможностями GPU. OpenCL разрабатывается и поддерживается некоммерческим консорциумом Khronos Group, в который входят много крупных компаний, включая Apple, AMD, Intel, nVidia, ARM, Sun Microsystems, Sony Computer Entertainment.

В лабораторной работе достигнуто ускорение программы на 30% с использованием векторизации, что показывает важность оптимизаций и использования профайлера при работе с GPU вычислениями.