Assignment 1 Report

By Sean Stephen

UOW ID 7311230

Report made on 11/05/2023

List of Files :

Common Files – common.h, common.cpp : Obtained from Tutorials

Task 1 Files – task1_7311230.cpp (adapted from tutorial1), task1.cl (given as part of assignment 1)

Task 2 Files – task2_7311230.cpp (adapted from tutorial3), task2.cl (given as part of assignment 1)

Task 3 Files – task3_7311230.cpp (adapted from tutorial1), arrayVec_mult.cl (self-written)

Explanation + Screenshots of Code

common.h

```
1   #pragma once
2   #ifndef _COMMON_H_
3   #define _COMMON_H_
4
5   #define CL_USE_DEPRECATED_OPENCL_2_0_APIS   // using OpenCL 1.2, some functions deprecated in OpenCL 2.0
6   #define __CL_ENABLE_EXCEPTIONS              // enable OpenCL exemptions
7
8   // C++ standard library and STL headers
9   #include <iostream>
10  #include <vector>
11  #include <sstream>
12  #include <fstream>
13
14  // OpenCL header, depending on OS
15  #ifdef __APPLE__
16  #include <OpenCL/cl.hpp>
17  #else
18  #include <CL/cl.hpp>
19  #endif
20
21  // function to handle error
22  void handle_error(cl::Error e);
23
24  // outputs message then quits
25  void quit_program(const std::string str);
26
27  // looks up and displays OpenCL error code as a string
28  const std::string lookup_error_code(cl_int error_code);
29
30  // allows the user to select a device, displays the available platform and device options
31  // returns whether selection was successful, the selected device and its platform
32  bool select_one_device(cl::Platform* platfm, cl::Device* dev);
33
34  // builds program from given filename
35  bool build_program(cl::Program* prog, const cl::Context* ctx, const std::string filename);
36
37  #endif
```

In this file, lines 2 and 3 are to define this header file once only.

Line 5 is to force the system to use OpenCL1.2 instead of the latest version.

Lines 15 – 19 determine if the user's device is MacOS or any other OS such as Windows, as this affects the header file to use as a wrapper for OpenCL C++.

The rest of the lines define the function prototypes for the functions implemented in common.cpp.

common.cpp

```
// allows the user to select a device, displays the available platform and device options
// returns whether selection was successful, the selected device and its platform
bool select_one_device(cl::Platform* platfm, cl::Device* dev)
{
    std::vector<cl::Platform> platforms;    // available platforms
    std::vector< std::vector<cl::Device> > platformDevices; // devices available for each platform
    std::string outputString;               // string for output
    unsigned int i, j;                      // counters

    try {
        // get the number of available OpenCL platforms
        cl::Platform::get(&platforms);
        std::cout << "Number of OpenCL platforms: " << platforms.size() << std::endl;

        // find and store the devices available to each platform
        for (i = 0; i < platforms.size(); i++)
        {
            std::vector<cl::Device> devices;        // available devices

            // get all devices available to the platform
            platforms[i].getDevices(CL_DEVICE_TYPE_ALL, &devices);

            // store available devices for the platform
            platformDevices.push_back(devices);
        }

        // display available platforms and devices
        std::cout << "--------------------" << std::endl;
        std::cout << "Available options:" << std::endl;

        // store options as platform and device indices
        std::vector< std::pair<int, int> > options;
        unsigned int optionCounter = 0; // option counter

        // for all platforms
        for (i = 0; i < platforms.size(); i++)
        {
            // for all devices per platform
            for (j = 0; j < platformDevices[i].size(); j++)
            {
                // display options
                std::cout << "Option " << optionCounter << ": Platform - ";

                // platform vendor name
                outputString = platforms[i].getInfo<CL_PLATFORM_VENDOR>();
                std::cout << outputString << ", Device - ";

                // device name
                outputString = platformDevices[i][j].getInfo<CL_DEVICE_NAME>();
                std::cout << outputString << std::endl;
```

The first function implementation is select_one_device. It receives a pointer to a platform object, and a device object and returns a Boolean (indicating successful selection of device). Its purpose is to initialise the platform and device objects that the user will choose.

The function uses a try-catch block to catch any OpenCL errors that bubble up.

From the top of the try block, the platforms object is first initialised with all the platforms available to the user's system.

Then in the following for-loop, a vector of each platform's devices is added to a vector, where each element corresponds to a platform's list of devices.

The next for-loop is a nested for-loop to display all devices for all platforms.

```
53
54                  // store option
55                  options.push_back(std::make_pair(i, j));
56                  optionCounter++; // increment option counter
57              }
58          }
59
60          std::cout << "\n--------------------" << std::endl;
61          std::cout << "Select a device: ";
62
63          std::string inputString;
64          unsigned int selectedOption;    // option that was selected
65
66          std::getline(std::cin, inputString);
67          std::istringstream stringStream(inputString);
68
69          // check whether valid option selected
70          // check if input was an integer
71          if (stringStream >> selectedOption)
72          {
73              char c;
74
75              // check if there was anything after the integer
76              if (!(stringStream >> c))
77              {
78                  // check if valid option range
79                  if (selectedOption >= 0 && selectedOption < optionCounter)
80                  {
81                      // return the platform and device
82                      int platformNumber = options[selectedOption].first;
83                      int deviceNumber = options[selectedOption].second;
84
85                      *platfm = platforms[platformNumber];
86                      *dev = platformDevices[platformNumber][deviceNumber];
87
88                      return true;
89                  }
90              }
91          }
92          // if invalid option selected
93          std::cout << "\n--------------------" << std::endl;
94          std::cout << "Invalid option." << std::endl;
95      }
96      // catch any OpenCL function errors
97      catch (cl::Error e) {
98          // call function to handle errors
99          handle_error(e);
100     }
101
102     return false;
103 }
```

The user is then prompted to enter which device (that corresponds to its platform) they desire, based on the index(denoted by optionCounter in the nested for-loop, which corresponds to the index of the pairs of indexes i, j, that exist in the vector of vectors of devices which is platformDevices) .

There is some input handling and the index pair is used to initialise the platform and device objects based on the index in the respective vectors.

```
105   // builds program from given filename
106   bool build_program(cl::Program* prog, const cl::Context* ctx, const std::string filename)
107   {
108       // get devices from the context
109       std::vector<cl::Device> contextDevices = ctx->getInfo<CL_CONTEXT_DEVICES>();
110
111       // open input file stream to .cl file
112       std::ifstream programFile(filename);
113
114       // check whether file was opened
115       if (!programFile.is_open())
116       {
117           std::cout << "File not found." << std::endl;
118           return false;
119       }
120
121       // create program string and load contents from the file
122       std::string programString(std::istreambuf_iterator<char>(programFile), (std::istreambuf_iterator<char>()));
123
124       // create program source from one input string
125       cl::Program::Sources source(1, std::make_pair(programString.c_str(), programString.length() + 1));
126       // create program from source
127       *prog = cl::Program(*ctx, source);
128
129       // try to build program
130       try {
131           // build the program for the devices in the context
132           prog->build(contextDevices);
133
134           std::cout << "Program build: Successful" << std::endl;
135           std::cout << "--------------------" << std::endl;
136       }
137       catch (cl::Error e) {
138           // if failed to build program
139           if (e.err() == CL_BUILD_PROGRAM_FAILURE)
140           {
141               // output program build log
142               std::cout << e.what() << ": Failed to build program." << std::endl;
143
144               // check build status for all all devices in context
145               for (unsigned int i = 0; i < contextDevices.size(); i++)
146               {
147                   // get device's program build status and check for error
148                   // if build error, output build log
149                   if (prog->getBuildInfo<CL_PROGRAM_BUILD_STATUS>(contextDevices[i]) == CL_BUILD_ERROR)
150                   {
151                       // get device name and build log
152                       std::string outputString = contextDevices[i].getInfo<CL_DEVICE_NAME>();
153                       std::string build_log = prog->getBuildInfo<CL_PROGRAM_BUILD_LOG>(contextDevices[i]);
154
155                       std::cout << "Device - " << outputString << ", build log:" << std::endl;
156                       st                                    << "--------------------" << std::endl;
157                   }
158               }
159
160               return false;
161           }
162           else
163           {
164               // call function to handle errors
165               handle_error(e);
166           }
167       }
168
169       return true;
170   }
171
```

{ } namespace std
Search Online

build_program is the next function implementation. It receives a program object pointer, a context object pointer and a string for the filename, returning a Boolean to indicate success. The purpose of this function is to build a program (initialise the program object), based on a given OpenCL file.

The function reads the input file into a string and then uses it to create a program source, then create the program with the source and context. Then it tries to build the program for the devices in the current context and any errors are caught and displayed in a verbose manner.

```
172   // function to handle error
173   void handle_error(cl::Error e)
174   {
175       // output OpenCL function that cause the error and the error code
176       std::cout << "Error in: " << e.what() << std::endl;
177       std::cout << "Error code: " << e.err() << " (" << lookup_error_code(e.err()) << ")" << std::endl;
178   }
179
```

handle_error receives an OpenCL error object and displays the error description and error code.

```
179
180   // function to quit program
181   void quit_program(const std::string str)
182   {
183       std::cout << str << std::endl;
184       std::cout << "Exiting the program..." << std::endl;
185
186   #ifdef _WIN32
187       // wait for a keypress on Windows OS before exiting
188       std::cout << "\npress a key to quit...";
189       std::cin.ignore();
190   #endif
191
192       exit(1);
193   }
194
```

quit_program is useful as it allows the program to quit early if there are errors in selecting the device or building the program.

```
195   // function to lookup and return error code string
196   const std::string lookup_error_code(cl_int error_code)
197   {
198       // look up error codes as defined in cl.hpp
199       switch (error_code) {
200       case CL_SUCCESS:
201           return "CL_SUCCESS";
202       case CL_DEVICE_NOT_FOUND:
203           return "CL_DEVICE_NOT_FOUND";
204       case CL_DEVICE_NOT_AVAILABLE:
205           return "CL_DEVICE_NOT_AVAILABLE";
206       case CL_COMPILER_NOT_AVAILABLE:
207           return "CL_COMPILER_NOT_AVAILABLE";
208       case CL_MEM_OBJECT_ALLOCATION_FAILURE:
209           return "CL_MEM_OBJECT_ALLOCATION_FAILURE";
210       case CL_OUT_OF_RESOURCES:
211           return "CL_OUT_OF_RESOURCES";
212       case CL_OUT_OF_HOST_MEMORY:
213           return "CL_OUT_OF_HOST_MEMORY";
214       case CL_PROFILING_INFO_NOT_AVAILABLE:
215           return "CL_PROFILING_INFO_NOT_AVAILABLE";
216       case CL_MEM_COPY_OVERLAP:
217           return "CL_MEM_COPY_OVERLAP";
218       case CL_IMAGE_FORMAT_MISMATCH:
219           return "CL_IMAGE_FORMAT_MISMATCH";
```

lookup_error_code is a very long function that receives an OpenCL integer and matches the correct error code, then returns a string version of it. At the end it also checks if the system is a MacOS system as there are some error codes not defined in MacOS's OpenCL C++ wrapper.

task1_7311230.cpp

The first 38 lines are defines, includes and the start of the main method where we declare objects that we need to use later on.

```
38
39  try {
40      // get all available OpenCL platforms
41      cl::Platform::get(&platforms);
42
43      std::cout << "\nThere are " << platforms.size() << " platforms on your system." << std::endl;
44      std::cout << "Here are their information : " << std::endl;
45
46      for (i = 0; i < platforms.size(); i++)
47      {
48          // output the index + 1  for better readability
49          std::cout << "\n\tPlatform " << i + 1 << std::endl;
50
51          // platform name
52          platforms[i].getInfo(CL_PLATFORM_NAME, &outputString);
53          std::cout << "\tPlatform Name : " << outputString << std::endl;
54
55          // get and output platform vendor name
56          outputString = platforms[i].getInfo<CL_PLATFORM_VENDOR>();
57          std::cout << "\tVendor: " << outputString << std::endl;
58
59          // get and output OpenCL version supported by the platform
60          outputString = platforms[i].getInfo<CL_PLATFORM_VERSION>();
61          std::cout << "\tVersion: " << outputString << std::endl;
62
63      }
64      std::cout << "\nPlease input the corresponding platform you would like to select : ";
65
66      std::cin >> platformNum;
67      std::cin.ignore(100, '\n');
68
69      std::cout << "\nYou have selected platform " << platformNum << std::endl;
70
71      // we only want GPU or CPU
72      platforms[platformNum - 1].getDevices(CL_DEVICE_TYPE_CPU | CL_DEVICE_TYPE_GPU, &tempDevices);
73
74      std::cout << "\nThere are " << tempDevices.size() << " devices on your platform." << std::endl;
75      std::cout << "Here are their information : " << std::endl;
76
77      for (i = 0; i < tempDevices.size(); i++)
78      {
79          // output the index + 1  for better readability
80          std::cout << "\n\tDevice " << i + 1 << std::endl;
81
82          // Device name
83          tempDevices[i].getInfo(CL_DEVICE_NAME, & outputString);
84          std::cout << "\tDevice Name : " << outputString << std::endl;
85      }
86
87      std::cout << "\nPlease input the corresponding device you would like to select : ";
88
89      std::cin >> deviceNum;
90      std::cin.ignore(100, '\n');
91
92      std::cout << "\nYou have selected device " << deviceNum << std::endl;
93
```

On line 39 we begin our try-catch block. Then we initialise all available platform objects in its vector and display some basic information. Since the first requirement of task 1 is to allow the user to select a device, we must then allow the user to first see all platforms and select one, then select a device under that platform. Once the platform is selected based on its index, we initialise all device objects in its vector and again allow the user to choose which device they want.

```
94          std::cout << "Here is its information :" << std::endl;
95
96
97          // platform name
98          platforms[platformNum - 1].getInfo(CL_PLATFORM_NAME, &outputString);
99          std::cout << "\n\tPlatform Name : " << outputString << std::endl;
100
101         // device type
102         cl_device_type devType;
103         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_TYPE, &devType);
104         if (devType == CL_DEVICE_TYPE_CPU)
105         {
106             outputString = "CL_DEVICE_TYPE _CPU";
107         }
108         else
109         {
110             outputString = "CL_DEVICE_TYPE _GPU";
111         }
112         std::cout << "\tDevice Type : " << outputString << std::endl;
113
114         // device name - returns string
115         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_NAME, &outputString);
116         std::cout << "\tDevice Name : " << outputString << std::endl;
117
118         // number of compute units - returns cl_uint
119         cl_uint maxCompUnits;
120         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_MAX_COMPUTE_UNITS, &maxCompUnits);
121         std::cout << "\tNumber of compute units : " << maxCompUnits << std::endl;
122
123         // max work group size - returns size_t
124         size_t maxWorkGSize;
125         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_MAX_WORK_GROUP_SIZE, &maxWorkGSize);
126         std::cout << "\tMaximum Work Group Size : " << maxWorkGSize << std::endl;
127
128         // max work item dimensions - returns cl_uint
129         cl_uint maxWorkDimensions;
130         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS, &maxWorkDimensions);
131         std::cout << "\tMaximum Work Item Dimensions : " << maxWorkDimensions << std::endl;
132
133
134         // max work item sizes - returns a vector of size_t
135         std::vector<size_t> work_item_sizes;
136         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_MAX_WORK_ITEM_SIZES, &work_item_sizes);
137         std::cout << "\tMaximum Work Item Sizes :";
138         for (i = 0; i < work_item_sizes.size(); i++)
139         {
140             std::cout << "\n\t\t\t\t\t\tDimension " << i + 1 << ": " << work_item_sizes[i];
141         }
142
143         // preferred vector width (int) - returns cl_uint
144         cl_uint prefVecWidth;
145         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT, &prefVecWidth);
146         std::cout << "\n\tPreferred vector width for integers : " << prefVecWidth << std::endl;
147
148         // local memory size - returns cl_ulong
149         cl_ulong mem_size;
150         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_LOCAL_MEM_SIZE, &mem_size);
151         std::cout << "\tLocal Memory Size : " << mem_size << std::endl;
152
153         std::cout << "\n-------------------------------------------------\n" << std::endl;
```

Once we have our selected device, we display all needed information, abusing the wrapper getInfo method for both the selected platform and device objects. Take note that not all calls to getInfo returns strings. Therefore I had to refer to https://man.opencl.org/clGetDeviceInfo.html to determine the return type of each flag, then provide the wrapper function with a pointer to the appropriate type. I do realise that the common.cpp has an implementation for selecting a device, but I wanted to try it out on my own.

```
154
155         // creating context with device
156         context = cl::Context(tempDevices[deviceNum - 1]);
157
158         // creating command queue with context and device
159         queue = cl::CommandQueue(context, tempDevices[deviceNum - 1]);
160
161         std::cout << "Context and Command Queue created! ~~~" << std::endl;
162
```

The next requirement was to create a context and a command queue. The context is first created with the selected device, then the command queue is created based on the context and device selected.

```
163         // check whether device supports extensions
164         // returns char[] delimited by spaces. use strtok to read through all, declare if found
165
166         bool fp16 = false, fp64 = false, icd = false;
167
168         char allExt[2000];
169         std::vector<char> allExtensions;
170         tempDevices[deviceNum - 1].getInfo(CL_DEVICE_EXTENSIONS, &allExt);
171
172     //  for (i = 0; i < 2000; i++)
173     //  {
174     //      std::cout << allExt[i];
175     //  }
176
177         // converting to string because I want to use getline() to split the char[] to the different substrings within
178         // tried to use strtok but im not sure why I could not get it to work
179
180         // manually null-terminating allExt to prevent buffer overrun error
181         allExt[2000 - 1] = '\0';
182
183         std::string allExtString(allExt);
184
185         std::stringstream sstream(allExtString);
186
187         std::string tempString;
188
189         std::vector<std::string> allExtStringVec;
190
191         while (std::getline(sstream, tempString, ' '))
192         {
193             allExtStringVec.push_back(tempString);
194         }
195
196         // looping through vector of strings to check if the needed extensions are present
197
198         for (i = 0; i < allExtStringVec.size(); i++)
199         {
200             if (allExtStringVec[i] == "cl_khr_fp16")
201             {
202                 fp16 = true;
203             }
204             else if (allExtStringVec[i] == "cl_khr_fp64")
205             {
206                 fp64 = true;
207             }
208             else if (allExtStringVec[i] == "cl_khr_icd")
209             {
210                 icd = true;
211             }
212         }
213
214         std::cout << "\nThe following extensions are / are not supported by the device :" << std::endl;
215         std::cout << "\tcl_khr_fp16 : " << (fp16 ? "Supported" : "Not Supported") << std::endl;
216         std::cout << "\tcl_khr_fp64 : " << (fp64 ? "Supported" : "Not Supported") << std::endl;
217         std::cout << "\tcl_khr_icd  : " << (icd ? "Supported" : "Not Supported") << std::endl;
218
219
```

The next requirement was to check if the selected device supported certain extensions. Since the getInfo method required a character array to store the information, but I am not sure of the exact size, I believe 2000 characters is enough to store all the information. Note that I manually null-terminated the char array as my IDE gave a warning that the last character might not be a null-termination, which might cause a buffer overrun.

Since the return of the getInfo is a char array where each extension name is delimited by a space, I tried to find ways to split the names into strings. I found that I could use a stringstream object and use getline to find delimeters, then store them in a string, which I can add to a vector. I had to initialise the stringstream object with a string therefore I initialised the string with the character array which performs the c-style string to string conversion for me. After all that, I looped through the vector of extension names and checked if the requested extensions were present, assigning true to the respective Booleans if found. Then I displayed if the extensions were supported by the device by using ternary operators with the Booleans.

```
220         // building program
221         if (!build_program(&program, &context, "task1.cl"))
222         {
223             // if OpenCL program build error
224             quit_program("OpenCL program build error.");
225         }
226
227         // display build logs
228         std::string buildLogs = program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(tempDevices[deviceNum - 1]);
229         std::cout << "\nBuild Logs :" << std::endl;
230         std::cout << (buildLogs.size() > 0 ? buildLogs : "No Build Logs were generated.") << std::endl;
231
```

The next requirement was to read the program source code from the provided task1.cl file and build the program, displaying the success and build logs (if any). I used the provided function build_program in common.cpp to build the program with the provided file. Then I used getBuildInfo on the created program to initialise a string for the build log and displayed it if there was any.

The last requirement was to find and display the number of kernels in the program, then create the kernels and display all their names. The createKernels function creates all kernels in the program at once, as opposed to creating each kernel object manually. This is useful when there are many kernels and the numbers are always changing, removing hard-coding from the equation. From here, the number of kernels is found with the size of the containing vector. We can then loop through each kernel object in the vector and use getInfo with the CL_KERNEL_FUNCTION_NAME flag to return a string which we can output the kernel's name with.

Task 1 Result in Terminal

```
There are 1 platforms on your system.
Here are their information :

        Platform 1
        Platform Name : AMD Accelerated Parallel Processing
        Vendor: Advanced Micro Devices, Inc.
        Version: OpenCL 2.1 AMD-APP (3516.0)

Please input the corresponding platform you would like to select : 1

You have selected platform 1

There are 1 devices on your platform.
Here are their information :

        Device 1
        Device Name : gfx1010:xnack-

Please input the corresponding device you would like to select : 1

You have selected device 1
Here is its information :

        Platform Name : AMD Accelerated Parallel Processing
        Device Type : CL_DEVICE_TYPE _GPU
        Device Name : gfx1010:xnack-
        Number of compute units : 20
        Maximum Work Group Size : 256
        Maximum Work Item Dimensions : 3
        Maximum Work Item Sizes :
                                        Dimension 1: 1024
                                        Dimension 2: 1024
                                        Dimension 3: 1024
        Preferred vector width for integers : 1
        Local Memory Size : 65536

-----------------------------------------------

Context and Command Queue created! ~~~

The following extensions are / are not supported by the device :
        cl_khr_fp16 : Supported
        cl_khr_fp64 : Supported
        cl_khr_icd  : Not Supported


Program build: Successful
--------------------

Build Logs :
C:\Users\User\AppData\Local\Temp\comgr-4e8024\input\CompileSource:29:1: warning: null character ignored [-Wnull-character]
<U+0000>
^
1 warning generated.


Number of Kernels : 5

All Kernel Names :

                add
                copy
                div
                mult
                sub

End of Assignment 1 Task 1

press a key to quit...
```

task2_7311230.cpp

Skipping explanation of all lines until line 48, where it is explained later.

```
46      // create and initialise our vector of alphabets
47
48      std::vector<cl_uchar> alphVec;
49
50      // we can make use of the ASCII table to initialise our vector easily
51      // z-a in ASCII is 122 to 97
52      // Z-A in ASCII is 90 to 65
53
54      cl_int i = 0;
55
56      for (i = 122; i >= 97; i--)
57      {
58          alphVec.push_back(i);
59      }
60
61      for (i = 90; i >= 65; i--)
62      {
63          alphVec.push_back(i);
64      }
65
66      // create and initialise our vector of unsigned ints
67
68      std::vector<cl_uint> uIntVec;
69
70      for (i = 0; i <= 1023; i++)
71      {
72          uIntVec.push_back(i);
73      }
74
75
```

Here we fulfil the first requirement by creating and initialising a vector of OpenCL unsigned characters with z-a then Z-A. I am making use of the ASCII values of these characters to initialise the vector with OpenCL integers in a for-loop.

Then I create and initalise the vector of 1024 unsigned integers with a for-loop.

```
76      // creating our three OpenCL buffer objects ,and initialising alphBuffer
77
78      // alphabet buffer
79      cl::Buffer alphBuffer;
80      alphBuffer = cl::Buffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(cl_uchar) * alphVec.size(), &alphVec[0]);
81
82      // unsigned char buffer
83      cl::Buffer uCharBuffer;
84      uCharBuffer = cl::Buffer(context, CL_MEM_WRITE_ONLY, sizeof(cl_uchar) * 52);
85
86      // unsigned int buffer
87      cl::Buffer uIntBuffer;
88      uIntBuffer = cl::Buffer(context, CL_MEM_READ_WRITE, sizeof(cl_uint) * 1024);
89
```

The next requirement is fulfilled here, by creating buffer objects with the requested flags of read only, write only or read and write. alphBuffer is also created using the copy host pointer flag and initalised with the unsigned character vector we created earlier. Buffers require the read/write flags and the size of the required buffer so that the memory space can be reserved.

```
90
91          // enqueue two OpenCL commands
92
93          // copy first buffer into second buffer
94          queue.enqueueCopyBuffer(alphBuffer, uCharBuffer, 0, 0, sizeof(cl_uchar) * 52);
95
96          queue.enqueueWriteBuffer(uIntBuffer, CL_TRUE, 0, sizeof(cl_uint) * 1024, &uIntVec[0]);
97
```

The following requirement is fulfilled by using enqueueCopy/WriteBuffer methods. I am using this as it is simpler than the mapping methods. The copy buffer method requires the source & destination buffers, the offset and frames and the size to copy. The write buffer requires the destination buffer, a flag whether it is blocking or not (blocks the program from proceeding until the write is completed), the offset, size to copy and the source vector.

```
2       cl::Platform platform;          // device's platform
3       cl::Device device;              // device used
4       cl::Context context;            // context for the device
5       cl::Program program;            // OpenCL program object
6       cl::Kernel kernel;              // a single kernel object
7       cl::CommandQueue queue;         // commandqueue for a context and device
8
9
0       try {
1           // select an OpenCL device
2           if (!select_one_device(&platform, &device))
3           {
4               // if no device selected
5               quit_program("Device not selected.");
6           }
7
8           // create a context from device
9           context = cl::Context(device);
0
1
2           // create command queue
3           queue = cl::CommandQueue(context, device);
```

```
97
98
99          // build the program
100         if (!build_program(&program, &context, "task2.cl"))
101         {
102             // if OpenCL program build error
103             quit_program("OpenCL program build error.");
104         }
105
106         kernel = cl::Kernel(program, "task2");
107
```

These are the steps (that make use of functions in common.cpp) that allow the user to select the device, create a context and command queue, then create a program and kernel for the given file and kernel names. Note that all objects up till the command queue were created at the start as the queue is needed to enqueue the copy/write buffers, and the buffers need too be created based on a context.

```
107
108         kernel.setArg(0, 12.45f);
109         kernel.setArg(1, uCharBuffer);
110         kernel.setArg(2, uIntBuffer);
111
112         std::cout << "\n\tEnqueue-ing kernel ...\n" << std::endl;
113
114         std::cout << "\t." << std::endl;
115         std::cout << "\t." << std::endl;
116         std::cout << "\t." << std::endl;
117         queue.enqueueTask(kernel);
118
119         std::cout << "\n\tReturned from kernel !!!\n" << std::endl;
120
```

setArg is used to set the arguments provided to the kernel and enqueueTask sends the kernel to be executed.

```
120
121         // read contents from buffers
122
123         std::cout << "Reading the contents of the two buffers now :" << std::endl;
124
125         // two placeholder buffers to read from
126         std::vector<cl_uchar> dispCharBuffer(52);
127         std::vector<cl_uint> dispIntBuffer(1024);
128
129         queue.enqueueReadBuffer(uCharBuffer, CL_TRUE, 0, sizeof(cl_uchar) * 52, &dispCharBuffer[0]);
130
131         std::cout << "\nDone reading Second Buffer. These are its contents in a 4 * 13 matrix :\n" << std::endl;
132
133         for ( i = 0; i < 4; i++)
134         {
135             for (int j = 0; j < 13; j++)
136             {
137                 std::cout << dispCharBuffer[(j * 1) + (i * 13)] << " ";
138             }
139             std::cout << std::endl;
140         }
141
142
143         queue.enqueueReadBuffer(uIntBuffer, CL_TRUE, 0, sizeof(cl_uint) * 1024, &dispIntBuffer[0]);
144
145         std::cout << "\nDone reading Third Buffer. These are its contents in a 64 * 16 matrix:\n" << std::endl;
146
147
148         for (i = 0; i < 64; i++)
149         {
150             for (int j = 0; j < 16; j++)
151             {
152                 std::cout << std::setw(4) << dispIntBuffer[(j * 1) + (i * 16)] << " ";
153             }
154             std::cout << std::endl;
155         }
156
157
158
159     }
```

The contents of the two buffers are read using enqueueReadBuffer, into vectors of the corresponding lengths and types. The method requires a source buffer, CL_TRUE or CL_FALSE to indicate blocking, offset, required size to read, as well as the destination vector.

Note that I used a nested for-loop so that I could display the information from the vectors in a 2D format. Accessing the element using j + i * row offset also generated a warning from my IDE asking to cast j before adding to i, as they had different sizes. I assume this will be different for different systems.

Task 2 Result in Terminal

```
Number of OpenCL platforms: 1
-------------------
Available options:
Option 0: Platform - Advanced Micro Devices, Inc., Device - gfx1010:xnack-

-------------------
Select a device: 0
Program build: Successful
-------------------

        Enqueue-ing kernel ...

        .
        .
        .

        Returned from kernel !!!

Reading the contents of the two buffers now :

Done reading Second Buffer. These are its contents in a 4 * 13 matrix :

z y x w v u t s r q p o n
m l k j i h g f e d c b a
Z Y X W V U T S R Q P O N
M L K J I H G F E D C B A

Done reading Third Buffer. These are its contents in a 64 * 16 matrix:

   0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
  16   17   18   19   20   21   22   23   24   25   26   27   28   29   30   31
  32   33   34   35   36   37   38   39   40   41   42   43   44   45   46   47
  48   49   50   51   52   53   54   55   56   57   58   59   60   61   62   63
  64   65   66   67   68   69   70   71   72   73   74   75   76   77   78   79
  80   81   82   83   84   85   86   87   88   89   90   91   92   93   94   95
  96   97   98   99  100  101  102  103  104  105  106  107  108  109  110  111
 112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127
 128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143
 144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159
 160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175
 176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191
 192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207
 208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223
 224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239
 240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255
 256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271
 272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287
 288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303
 304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319
 320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335
 336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351
 352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367
 368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383
 384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399
 400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415
 416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431
 432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447
 448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463
 464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479
 480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495
 496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511
 512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527
 528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543
 544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559
 560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575
 576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591
 592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607
 608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623
 624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639
 640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655
 656  657  658  659  660  661  662  663  664  665  666  667  668  669  670  671
 672  673  674  675  676  677  678  679  680  681  682  683  684  685  686  687
 688  689  690  691  692  693  694  695  696  697  698  699  700  701  702  703
 704  705  706  707  708  709  710  711  712  713  714  715  716  717  718  719
 720  721  722  723  724  725  726  727  728  729  730  731  732  733  734  735
 736  737  738  739  740  741  742  743  744  745  746  747  748  749  750  751
 752  753  754  755  756  757  758  759  760  761  762  763  764  765  766  767
 768  769  770  771  772  773  774  775  776  777  778  779  780  781  782  783
 784  785  786  787  788  789  790  791  792  793  794  795  796  797  798  799
 800  801  802  803  804  805  806  807  808  809  810  811  812  813  814  815
 816  817  818  819  820  821  822  823  824  825  826  827  828  829  830  831
 832  833  834  835  836  837  838  839  840  841  842  843  844  845  846  847
 848  849  850  851  852  853  854  855  856  857  858  859  860  861  862  863
 864  865  866  867  868  869  870  871  872  873  874  875  876  877  878  879
 880  881  882  883  884  885  886  887  888  889  890  891  892  893  894  895
 896  897  898  899  900  901  902  903  904  905  906  907  908  909  910  911
 912  913  914  915  916  917  918  919  920  921  922  923  924  925  926  927
 928  929  930  931  932  933  934  935  936  937  938  939  940  941  942  943
 944  945  946  947  948  949  950  951  952  953  954  955  956  957  958  959
 960  961  962  963  964  965  966  967  968  969  970  971  972  973  974  975
 976  977  978  979  980  981  982  983  984  985  986  987  988  989  990  991
 992  993  994  995  996  997  998  999 1000 1001 1002 1003 1004 1005 1006 1007
1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023

press a key to quit...
```

task3_7311230.cpp

```
49          // build the program
50          if(!build_program(&program, &context, "arrayVec_mult.cl"))
51          {
52              // if OpenCL program build error
53              quit_program("OpenCL program build error.");
54          }
55
56          // create a kernel
57          kernel = cl::Kernel(program, "arrayVec_mult");
58
59          // create command queue
60          queue = cl::CommandQueue(context, device);
61
```

Here we build the program, providing the filename of our self-written kernel file, then build a single kernel by specifying the kernel name. All previous lines are standard steps to set-up the

```
61
62          // create buffer
63          addBuffer = cl::Buffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR, sizeof(cl_int) * LENGTH, &all3s[0]);
64
65          // get input from user
66
67          int numInput;
68          bool withinRange = false;
69
70          // check if the value is between 2 and 99 inclusive
71          while (!withinRange)
72          {
73              std::cout << "\nDear User, This program takes in a single number between 2 and 99 inclusive. Please enter a number : ";
74              std::cin >> numInput;
75              std::cin.ignore(100, '\n');
76
77              if (numInput >= 2 && numInput <= 99)
78              {
79                  withinRange = true;
80              }
81
82              else
83              {
84                  std::cout << "\nNumber not between 2 and 99 Inclusive. Please try again." << std::endl;
85              }
86          }
87
88
89          std::cout << "\nPlease note that you have entered : " << numInput << std::endl;
90
91          std::cout << "\nOriginal Array : \n" << std::endl;
92
93          for (int i = 0; i < 64; i++)
94          {
95              for (int j = 0; j < 8; j++)
96              {
97                  std::cout << std::setw(4) << all3s[j + i * 4] << " ";
98              }
99              std::cout << std::endl;
100         }
```

Since Task 3's requirements is to get a user's input where the number that is input will determine how much to increment the numbers in the 512 element vector, my design is to first create a vector with 512 elements, all initialised with the value 3. The increment starts after 3, therefore conceptually, we need to skip the first element, then add the input number * index in the vector (after accounting for offset of 1) to 3. This is easily done in a for-loop to get the correct index to multiply, but since we want to use our kernel to speed things up, we need to do it the abovementioned way.

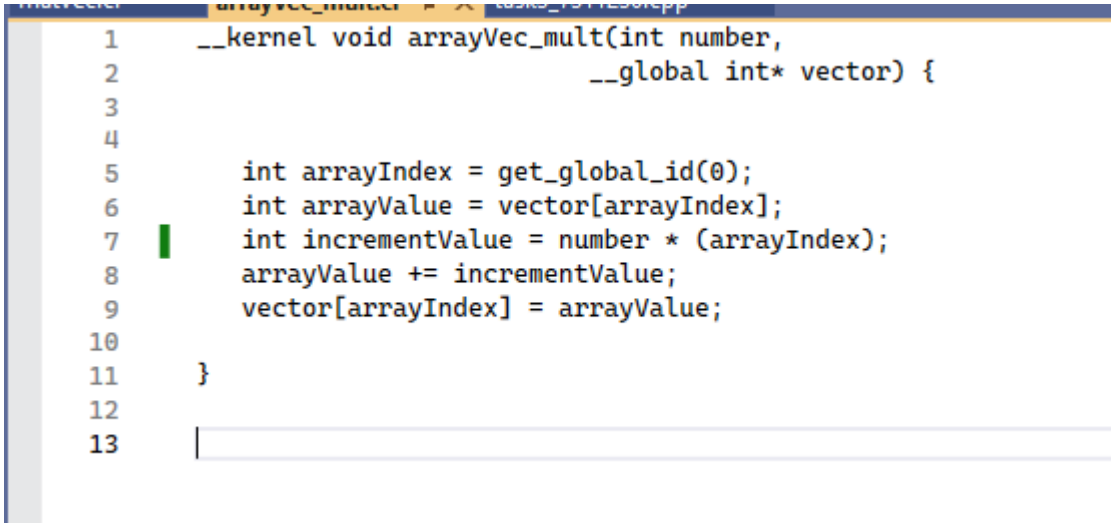
In the above code, I create the buffer where we will read the value of 3, write the result after the multiplication, then read the buffer back into a vector of size 512. Then I read the user's input, making sure it is within the 2-99 inclusive range stated in the requirements, and display the original vector before we enqueue the kernel for execution.

```
101
102         // set kernel arguments
103         kernel.setArg(0, numInput);
104         kernel.setArg(1, addBuffer);
105
106
107         // enqueue kernel for execution
108         //queue.enqueueTask(kernel);
109
110         // offset 1 because we want the first element to remain 1
111         // globalSize 512 because we want to use the index of the array to compute how much to increment
112         cl::NDRange offset(1);
113         cl::NDRange globalSize(512);
114
115         queue.enqueueNDRangeKernel(kernel, offset, globalSize);
116
117         std::cout << "\nKernel enqueued." << std::endl;
118         std::cout << "--------------------" << std::endl;
119
120         // enqueue command to read from device to host memory
121         queue.enqueueReadBuffer(addBuffer, CL_TRUE, 0, sizeof(cl_int) * LENGTH, &resultVec[0]);
122
123
124         std::cout << "\n Our Results : \n" << std::endl;
125
126         // check the results
127         for (int i = 0; i < 64; i++)
128         {
129             for (int j = 0; j < 8; j++)
130             {
131                 std::cout << std::setw(4) << resultVec[j + i * 4] << " ";
132             }
133             std::cout << std::endl;
134         }
135     }
136     // catch any OpenCL function errors
137     catch (cl::Error e) {
138         // call function to handle errors
139         handle_error(e);
140     }
```

Here, I first set the arguments for the kernel. Then I use enqueueNDRangeKernel to enqueue the kernel for execution. I can make use of the offset of 1 to skip the first element (the first 3). The globalSize is set to 512 so that in the kernel I can use the global id to determine the index of the element that is being accessed in that instance.

enqueueReadBuffer is used to read the contents of the buffer back into a vector, where it is displayed in a 2D format using a nested for-loop.

arrayVec_mult.cl



```
1   __kernel void arrayVec_mult(int number,
2                               __global int* vector) {
3
4
5       int arrayIndex = get_global_id(0);
6       int arrayValue = vector[arrayIndex];
7       int incrementValue = number * (arrayIndex);
8       arrayValue += incrementValue;
9       vector[arrayIndex] = arrayValue;
10
11  }
12
13
```

This is the kernel code used in Task 3. The parameter is the integer that the user entered used for our incrementing, as well as the buffer where our elements are stored.

The global id is used to determine which index of the element we are accessing in that instance of the kernel. From there we can first get the original value of 3, determine the amount to increment by using the user entered number to multiply with the index. Then we add the original value of 3 with the increment and reassign that element with the result value.

Task 3 Result in Terminal

```
Number of OpenCL platforms: 1
--------------------
Available options:
Option 0: Platform - Advanced Micro Devices, Inc., Device - gfx1010:xnack-

--------------------
Select a device: 0
Program build: Successful
--------------------

Dear User, This program takes in a single number between 2 and 99 inclusive. Please enter a number : 3

Please note that you have entered : 3

Original Array :

   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
   3    3    3    3    3    3    3    3
```

```
Kernel enqueued.
--------------------

 Our Results :

   3    6    9   12   15   18   21   24
  15   18   21   24   27   30   33   36
  27   30   33   36   39   42   45   48
  39   42   45   48   51   54   57   60
  51   54   57   60   63   66   69   72
  63   66   69   72   75   78   81   84
  75   78   81   84   87   90   93   96
  87   90   93   96   99  102  105  108
  99  102  105  108  111  114  117  120
 111  114  117  120  123  126  129  132
 123  126  129  132  135  138  141  144
 135  138  141  144  147  150  153  156
 147  150  153  156  159  162  165  168
 159  162  165  168  171  174  177  180
 171  174  177  180  183  186  189  192
 183  186  189  192  195  198  201  204
 195  198  201  204  207  210  213  216
 207  210  213  216  219  222  225  228
 219  222  225  228  231  234  237  240
 231  234  237  240  243  246  249  252
 243  246  249  252  255  258  261  264
 255  258  261  264  267  270  273  276
 267  270  273  276  279  282  285  288
 279  282  285  288  291  294  297  300
 291  294  297  300  303  306  309  312
 303  306  309  312  315  318  321  324
 315  318  321  324  327  330  333  336
 327  330  333  336  339  342  345  348
 339  342  345  348  351  354  357  360
 351  354  357  360  363  366  369  372
 363  366  369  372  375  378  381  384
 375  378  381  384  387  390  393  396
 387  390  393  396  399  402  405  408
 399  402  405  408  411  414  417  420
 411  414  417  420  423  426  429  432
 423  426  429  432  435  438  441  444
 435  438  441  444  447  450  453  456
 447  450  453  456  459  462  465  468
 459  462  465  468  471  474  477  480
 471  474  477  480  483  486  489  492
 483  486  489  492  495  498  501  504
 495  498  501  504  507  510  513  516
 507  510  513  516  519  522  525  528
 519  522  525  528  531  534  537  540
 531  534  537  540  543  546  549  552
 543  546  549  552  555  558  561  564
 555  558  561  564  567  570  573  576
 567  570  573  576  579  582  585  588
 579  582  585  588  591  594  597  600
 591  594  597  600  603  606  609  612
 603  606  609  612  615  618  621  624
 615  618  621  624  627  630  633  636
 627  630  633  636  639  642  645  648
 639  642  645  648  651  654  657  660
 651  654  657  660  663  666  669  672
 663  666  669  672  675  678  681  684
 675  678  681  684  687  690  693  696
 687  690  693  696  699  702  705  708
 699  702  705  708  711  714  717  720
 711  714  717  720  723  726  729  732
 723  726  729  732  735  738  741  744
 735  738  741  744  747  750  753  756
 747  750  753  756  759  762  765  768
 759  762  765  768  771  774  777  780

press a key to quit...
```