Computational Science on Many-Core Architectures: Exercise 8:

Viktor Beck, 11713110

December 2022

Dot Product with OpenCL

1)

Here, we use the vector_add.cpp file as starting point for the dot product by simply converting the OpenCL kernel into a vector multiplication kernel (x[i] = x[i] * y[i]). After the kernel call in the main function we sum the entries of vector x on the CPU - see Appendix. Note, that the code does only produce correct results when `reps = 1` (repetitions for measuring the execution time). But since the time is not dependent on the content of the vector it does not matter.

2) and 3)

For the performance comparison between CUDA and OpenCL an old CUDA code from exercise 2 was used. One can see that the CUDA implementation performs way better than the OpenCL implementation (... maybe because my CUDA dot product implementation is so good). Also, one can see the performance of the CPU which is clearly the worst. To get the CPU instead of the GPU running we simply have to exchange the 0 in the `platform_ids[0]` with 1 in the last line of the code below.

```
//
// Query platform:
//
cl_uint num_platforms;
cl_platform_id platform_ids[42];   //no more than 42 platforms supported...
err = clGetPlatformIDs(42, platform_ids, &num_platforms); OPENCL_ERR_CHECK(err);
std::cout << "# Platforms found: " << num_platforms << std::endl;
cl_platform_id my_platform = platform_ids[0];
```

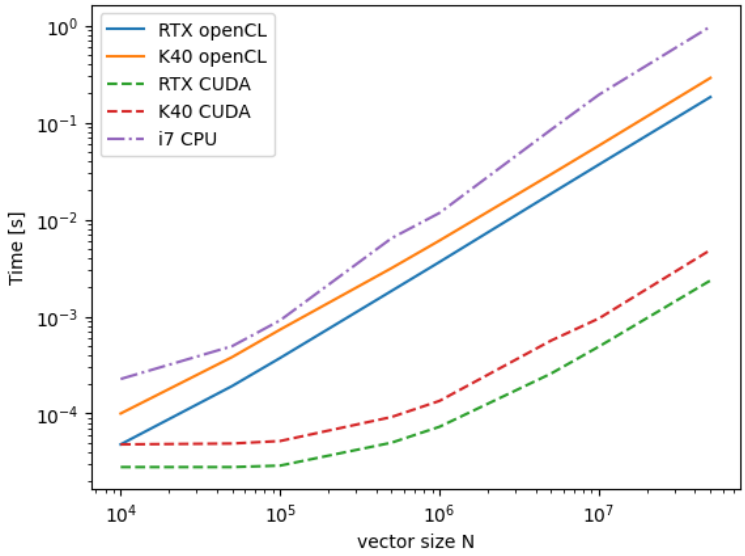

Figure 1: Execution times for different vector sizes of N for the dot product

Appendix

Sorry for the horrible layout of the code listing here. The code was also uploaded with the report.

Task 1),2),3)

```
//
// Tutorial for demonstrating a simple OpenCL vector addition kernel
//
// Author: Karl Rupp    rupp@iue.tuwien.ac.at
//

typedef double       ScalarType;


#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <stdexcept>

#ifdef __APPLE__
#include <OpenCL/cl.h>
#else
#include <CL/cl.h>
#endif

// Helper include file for error checking
#include "ocl-error.hpp"
#include "timer.hpp"


```

```
const char *my_opencl_program = ""
"#pragma OPENCL EXTENSION cl_khr_fp64 : enable\n"    // required to enable 'double' inside OpenCL pro
""
"__kernel void dot_product(__global double *x,\n"
"                          __global double *y,\n"
"                          unsigned int N\n)"
"{\n"
"  for (unsigned int i  = get_global_id(0);\n"
"                    i  < N;\n"
"                    i += get_global_size(0))\n"
"    x[i] = x[i] * y[i];\n"
"}";  // you can have multiple kernels within a single OpenCL program. For simplicity, this OpenCL pr


int main()
{
  cl_int err;

  //
  /////////////////////////// Part 1: Set up an OpenCL context with one device ////////////////////
  //

  //
  // Query platform:
  //
  cl_uint num_platforms;
  cl_platform_id platform_ids[42];   //no more than 42 platforms supported...
  err = clGetPlatformIDs(42, platform_ids, &num_platforms); OPENCL_ERR_CHECK(err);
  std::cout << "# Platforms found: " << num_platforms << std::endl;
  cl_platform_id my_platform = platform_ids[0];


  //
  // Query devices:
  //
  cl_device_id device_ids[42];
  cl_uint num_devices;
  err = clGetDeviceIDs(my_platform, CL_DEVICE_TYPE_ALL, 42, device_ids, &num_devices); OPENCL_ERR_CHE
  std::cout << "# Devices found: " << num_devices << std::endl;
  cl_device_id my_device_id = device_ids[0];

  char device_name[64];
  size_t device_name_len = 0;
  err = clGetDeviceInfo(my_device_id, CL_DEVICE_NAME, sizeof(char)*63, device_name, &device_name_len)
  std::cout << "Using the following device: " << device_name << std::endl;

  //
  // Create context:
  //
  cl_context my_context = clCreateContext(0, 1, &my_device_id, NULL, NULL, &err); OPENCL_ERR_CHECK(er


  //
  // create a command queue for the device:
  //
  cl_command_queue my_queue = clCreateCommandQueueWithProperties(my_context, my_device_id, 0, &err);



  //
  /////////////////////////// Part 2: Create a program and extract kernels ////////////////////////
  //
```

```
89
90    Timer timer;
91    timer.reset();
92
93    //
94    // Build the program:
95    //
96    size_t source_len = std::string(my_opencl_program).length();
97    cl_program prog = clCreateProgramWithSource(my_context, 1, &my_opencl_program, &source_len, &err);O
98    err = clBuildProgram(prog, 0, NULL, NULL, NULL, NULL);
99
100   //
101   // Print compiler errors if there was a problem:
102   //
103   if (err != CL_SUCCESS) {
104
105     char *build_log;
106     size_t ret_val_size;
107     err = clGetProgramBuildInfo(prog, my_device_id, CL_PROGRAM_BUILD_LOG, 0, NULL, &ret_val_size);
108     build_log = (char *)malloc(sizeof(char) * (ret_val_size+1));
109     err = clGetProgramBuildInfo(prog, my_device_id, CL_PROGRAM_BUILD_LOG, ret_val_size, build_log, NU
110     build_log[ret_val_size] = '\0'; // terminate string
111     std::cout << "Log: " << build_log << std::endl;
112     free(build_log);
113     std::cout << "OpenCL program sources: " << std::endl << my_opencl_program << std::endl;
114     return EXIT_FAILURE;
115   }
116
117   //
118   // Extract the only kernel in the program:
119   //
120   cl_kernel my_kernel = clCreateKernel(prog, "dot_product", &err); OPENCL_ERR_CHECK(err);
121
122   std::cout << "Time to compile and create kernel: " << timer.get() << std::endl;
123
124
125   //
126   /////////////////////////// Part 3: Create memory buffers ////////////////////////////////
127   //
128
129   //
130   // Set up buffers on host:
131   //
132   cl_uint vector_size = 50000000;
133   std::vector<ScalarType> x(vector_size, 3.0);
134   std::vector<ScalarType> y(vector_size, 2.0);
135
136   std::cout << std::endl;
137   std::cout << "Vectors before kernel launch:" << std::endl;
138   std::cout << "x: " << x[0] << " " << x[1] << " " << x[2] << " ..." << std::endl;
139   std::cout << "y: " << y[0] << " " << y[1] << " " << y[2] << " ..." << std::endl;
140
141   //
142   // Now set up OpenCL buffers:
143   //
144   cl_mem ocl_x = clCreateBuffer(my_context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR, vector_size * s
145   cl_mem ocl_y = clCreateBuffer(my_context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR, vector_size * s
146
147
148   //
149   /////////////////////////// Part 4: Run kernel ///////////////////////////////////
150   //
```

```
151   size_t  local_size = 128;
152   size_t global_size = 128*128;
153
154
155   //
156   // Set kernel arguments:
157   //
158   err = clSetKernelArg(my_kernel, 0, sizeof(cl_mem),  (void*)&ocl_x); OPENCL_ERR_CHECK(err);
159   err = clSetKernelArg(my_kernel, 1, sizeof(cl_mem),  (void*)&ocl_y); OPENCL_ERR_CHECK(err);
160   err = clSetKernelArg(my_kernel, 2, sizeof(cl_uint), (void*)&vector_size); OPENCL_ERR_CHECK(err);
161
162   std::vector<double> times;
163   int reps = 6;
164   double sum;
165
166   // START TIMING HERE:
167
168   for (int i=0; i<reps; i++){
169     timer.reset();
170
171     //
172     // Enqueue kernel in command queue:
173     //
174     err = clEnqueueNDRangeKernel(my_queue, my_kernel, 1, NULL, &global_size, &local_size, 0, NULL, NU
175
176     // wait for all operations in queue to finish:
177     err = clFinish(my_queue); OPENCL_ERR_CHECK(err);
178
179
180     //
181     ////////////////////////// Part 5: Get data from OpenCL buffer ////////////////////////////////
182     //
183
184     err = clEnqueueReadBuffer(my_queue, ocl_x, CL_TRUE, 0, sizeof(ScalarType) * x.size(), &(x[0]), 0,
185
186     // summing like a pro on the CPU
187     sum = 0;
188     for (int i = 0; i < vector_size; i++){
189       sum += x[i];
190     }
191     times.push_back(timer.get());
192   }
193
194   std::cout << "Exec. time\n" << times[reps/2] << std::endl;
195
196   // TIMER END
197
198   std::cout << std::endl;
199   std::cout << "Vectors after kernel execution:" << std::endl;
200   std::cout << "x: " << x[0] << " " << x[1] << " " << x[2] << " ..." << std::endl;
201   std::cout << "y: " << y[0] << " " << y[1] << " " << y[2] << " ..." << std::endl;
202
203
204   std::cout << "Dot product of x and y = " << sum << std::endl;
205
206
207   //
208   // cleanup
209   //
210   clReleaseMemObject(ocl_x);
211   clReleaseMemObject(ocl_y);
212   clReleaseProgram(prog);
```

```
213   clReleaseCommandQueue(my_queue);
214   clReleaseContext(my_context);
215
216   std::cout << std::endl;
217   std::cout << "#" << std::endl;
218   std::cout << "# My first OpenCL application finished successfully!" << std::endl;
219   std::cout << "#" << std::endl;
220   return EXIT_SUCCESS;
221 }
```