November 15, 2021 at 08:58

1. Introduction. 이 문서는 OpenCL을 이용한 병렬 연산을 지원하는 라이브러리를 설명한다.

2. Declaration.

⟨ mpoi.h 2 ⟩ ≡

```
#ifndef __MULTI_PROCESSING_OBJECT_INTERFACE_H_
#define __MULTI_PROCESSING_OBJECT_INTERFACE_H_
#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif
#include <vector>
#include <map>
#include <iostream>
#include <fstream>
#include <iterator>
  using namespace std;
  class mpoi {
    ⟨ Constants of mpoi 16 ⟩
    ⟨ Data members of mpoi 3 ⟩
    ⟨ Member functions of mpoi 6 ⟩
  };
#endif
```

3. OpenCL을 이용한 multi-processing interface를 정의하기 위하여 다음과 같은 멤버 변수들이 필요하다.

⟨ Data members of mpoi 3 ⟩ ≡

```
protected:
  cl_device_id _device_id;
  cl_context _context;
  cl_command_queue _cmd_queue;
  cl_program _program; vector < cl_kernel > _kernels; map < size_t , cl_mem > _buffers;
  size_t _next_key;
  string _src;
```

This code is used in section 2.

4. Implementation.

⟨ mpoi.cpp 4 ⟩ ≡

```
#include "mpoi.h"
  ⟨ Constructors and destructor of mpoi 12 ⟩
  ⟨ Methods to handle mpoi object 5 ⟩
  ⟨ Setup and cleanup of OpenCL 7 ⟩
  ⟨ Method to build a kernel 9 ⟩
  ⟨ Methods to display platform info 14 ⟩
  ⟨ Methods to handle device memory buffers 17 ⟩
  ⟨ Methods to set kernel arguments 21 ⟩
  ⟨ Methods to execute a kernel 23 ⟩
```

5. mpoi 객체를 다루기 위한 methods.

⟨ Methods to handle mpoi object 5 ⟩ ≡

```
mpoi &mpoi::operator=(const mpoi &obj) {
  _next_key = 0;
  _src = obj._src;
  _setup_opencl( );
  if (_src ≠ "") {
    build_program(_src);
  }
  return *this;
}
```

This code is used in section 4.

6. ⟨ Member functions of mpoi 6 ⟩ ≡

```
public:
  mpoi &operator=(const mpoi &);
```

See also sections 8, 10, 13, 15, 18, 20, 22, and 24.

This code is used in section 2.

7. OpenCL 사용을 위한 기본 설정과 사용 후 정리.

⟨ Setup and cleanup of OpenCL 7 ⟩ ≡

```
47  void mpoi::_setup_opencl() {
48    cl_uint num_platforms;
49    cl_int err = clGetPlatformIDs(0, Λ, &num_platforms);
50    if (err ≠ CL_SUCCESS ∨ num_platforms < 1) {
51      cerr ≪ "Failed␣to␣find␣any␣OpenCL␣platforms.\n";
52      exit(1);
53    }
54    cl_platform_id *platformIDs = (cl_platform_id *) new cl_platform_id[num_platforms];
55    err = clGetPlatformIDs(num_platforms, platformIDs, Λ);
56    if (err ≠ CL_SUCCESS) {
57      cerr ≪ "Failed␣to␣find␣any␣OpenCL␣platforms.\n";
58      exit(1);
59    }
60    cout ≪ num_platforms ≪ "␣OpenCL␣platform(s)␣found.\n";
61    bool device_found = false;
62    cl_uint current_max_compute_units = 0;
63    cl_uint max_compute_units = 0;
64    for (cl_uint i = 0; i ≠ num_platforms; i++) {
65      cout ≪ "Platform␣#␣" ≪ i ≪ endl;
66      cl_uint num_devices;
67      err = clGetDeviceIDs(platformIDs[i], CL_DEVICE_TYPE_GPU, 0, Λ, &num_devices);
68      if (num_devices < 1) {
69        cerr ≪ "No␣GPU␣devices␣found␣for␣platform␣" ≪ platformIDs[i] ≪ endl;
70      }
71      else {
72        device_found = true;
73        cout ≪ num_devices ≪ "␣GPU␣device(s)␣found␣for␣platform␣" ≪ platformIDs[i] ≪ endl;
74        cl_device_id *deviceIDs = (cl_device_id *) new cl_device_id[num_devices];
75        for (cl_uint j = 0; j ≠ num_devices; j++) {
76          err = clGetDeviceIDs(platformIDs[i], CL_DEVICE_TYPE_GPU, 1, &deviceIDs[j], Λ);
77          cl_uint device_vendor_id;
78          err = clGetDeviceInfo(deviceIDs[j], CL_DEVICE_VENDOR_ID, sizeof(cl_uint),
                &device_vendor_id, Λ);
79          cout ≪ "Device␣vendor␣ID:␣" ≪ device_vendor_id ≪ endl;
80          err = clGetDeviceInfo(deviceIDs[j], CL_DEVICE_MAX_COMPUTE_UNITS, sizeof(cl_uint),
                &max_compute_units, Λ);
81          cout ≪ "Device␣has␣" ≪ max_compute_units ≪ "␣compute␣units.\n";
82          if (max_compute_units > current_max_compute_units) {
83            current_max_compute_units = max_compute_units;
84            _device_id = deviceIDs[j];
85          }
86        }
87        delete[] deviceIDs;
88      }
89    }
90    if (¬device_found) {
91      cerr ≪ "No␣OpenCL␣GPU␣devices␣found␣through␣all␣the␣platforms.\n";
```

```
92          exit(1);
93        }
94        _context = clCreateContext(Λ, 1, &_device_id, Λ, Λ, &err);
95        if (err ≠ CL_SUCCESS) {
96          cerr ≪ "Error in creating a context.\n";
97          exit(1);
98        }
99        _cmd_queue = clCreateCommandQueue(_context, _device_id, 0, &err);
100       if (err ≠ CL_SUCCESS) {
101         cerr ≪ "Error in creating a command queue.\n";
102         exit(1);
103       }
104       _program = Λ;
105       delete[] platformIDs;
106     }
107     void mpoi::_cleanup_opencl( ) {
108       clFlush(_cmd_queue);
109       clFinish(_cmd_queue);
110       for (size_t i = 0; i ≠ _kernels.size( ); i++) {
111         clReleaseKernel(_kernels[i]);
112       }
113       if (¬_program) {
114         clReleaseProgram(_program);
115       }
116       clReleaseCommandQueue(_cmd_queue);
117       clReleaseContext(_context);
118     }
```

This code is used in section 4.

8. ⟨ Member functions of **mpoi** 6 ⟩ +≡

```
119  private:
120    void _setup_opencl( );
121    void _cleanup_opencl( );
```

9. Kernel을 생성하는 method.

⟨ Method to build a kernel 9 ⟩ ≡

```
122  void mpoi::build_program(const string &src_file) {
123    ifstream in(src_file);
124    if (in.is_open()) {
125      string src((istreambuf_iterator⟨char⟩(in)),istreambuf_iterator⟨char⟩());
126      const char *src_string = src.c_str();
127      const size_t src_length = src.length();
128      cl_int err;
129      _program = clCreateProgramWithSource(_context,1,(const char **) &src_string,(const size_t *)
           &src_length,&err);
130      if (err ≠ CL_SUCCESS) {
131        cerr ≪ "Error␣in␣creating␣a␣program.\n";
132      }
133      err = clBuildProgram(_program,1,&_device_id,Λ,Λ,Λ);
134      ⟨ Check kernel program build error 11 ⟩;
135    }
136    else {
137      exit(1);
138    }
139  }
140  size_t mpoi::create_kernel(const string &name) {
141    size_t id = _kernels.size();
142    cl_int err;
143    _kernels.push_back(clCreateKernel(_program,name.c_str(),&err));
144    return id;
145  }
```

This code is used in section 4.

10. ⟨ Member functions of **mpoi** 6 ⟩ +≡

```
146  public:
147    void build_program(const string &);
148    size_t create_kernel(const string &);
```

11. Kernel program의 build 결과를 확인하고, 오류가 있으면 build log를 출력한다.

⟨ Check kernel program build error 11 ⟩ ≡

```
149   if (err ≠ CL_SUCCESS) {
150     cerr ≪ "Error in building a program.\n";
151     cl_build_status build_status;
152     clGetProgramBuildInfo(_program, _device_id, CL_PROGRAM_BUILD_STATUS, sizeof
          (cl_build_status), &build_status, Λ);
153     if (build_status ≠ CL_SUCCESS) {
154       size_t ret_val_size;
155       clGetProgramBuildInfo(_program, _device_id, CL_PROGRAM_BUILD_LOG, 0, Λ, &ret_val_size);
156       char *build_log = (char *) new char[ret_val_size + 1];
157       clGetProgramBuildInfo(_program, _device_id, CL_PROGRAM_BUILD_LOG, ret_val_size, build_log, Λ);
158       build_log[ret_val_size] = '\0';
159       cerr ≪ "BUILD LOG: " ≪ build_log ≪ endl;
160       delete[] build_log;
161     }
162   }
```

This code is used in section 9.

12. 생성자와 소멸자.

⟨ Constructors and destructor of **mpoi** 12 ⟩ ≡

```
163   mpoi::mpoi()
164   : _next_key(0), _src("") {
165     _setup_opencl();
166   }
167   mpoi::mpoi(const string &src)
168   : _next_key(0), _src(src) {
169     _setup_opencl();
170     build_program(_src);
171   }
172   mpoi::mpoi(const mpoi &obj)
173   : _device_id(obj._device_id), _context(obj._context), _cmd_queue(obj._cmd_queue),
        _program(obj._program), _kernels(obj._kernels), _buffers(obj._buffers), _next_key(obj._next_key),
        _src(obj._src) { }
174   mpoi::~mpoi() {
175     _cleanup_opencl();
176   }
```

This code is used in section 4.

13. ⟨ Member functions of **mpoi** 6 ⟩ +≡

```
177  public:
178   mpoi();
179   mpoi(const string &);
180   mpoi(const mpoi &);
181   virtual ~mpoi();
```

14. OpenCL platform에 관한 정보를 보여주기 위한 methods.

```
⟨ Methods to display platform info 14 ⟩ ≡
  void mpoi::display_platform_info() const {
    cl_uint num_platforms;
    cl_int err = clGetPlatformIDs(0, Λ, &num_platforms);
    if (err ≠ CL_SUCCESS ∨ num_platforms < 1) {
      cerr ≪ "Failed to find any OpenCL platforms.\n";
      return;
    }
    cl_platform_id *platformIDs = (cl_platform_id *) new cl_platform_id[num_platforms];
    err = clGetPlatformIDs(num_platforms, platformIDs, Λ);
    if (err ≠ CL_SUCCESS) {
      cerr ≪ "Failed to find any OpenCL platforms.\n";
      return;
    }
    cout ≪ "Number of platforms:\t" ≪ num_platforms ≪ endl;
    for (cl_uint i = 0; i ≠ num_platforms; i++) {
      _display_platform_info(platformIDs[i], CL_PLATFORM_PROFILE, "CL_PLATFORM_PROFILE   ");
      _display_platform_info(platformIDs[i], CL_PLATFORM_VERSION, "CL_PLATFORM_VERSION   ");
      _display_platform_info(platformIDs[i], CL_PLATFORM_VENDOR, "CL_PLATFORM_VENDOR     ");
      _display_platform_info(platformIDs[i], CL_PLATFORM_EXTENSIONS, "CL_PLATFORM_EXTENSIONS");
    }
    delete[] platformIDs;
  }
  void mpoi::_display_platform_info(cl_platform_id id, cl_platform_info name, string str) const {
    size_t param_value_size;
    cl_int err = clGetPlatformInfo(id, name, 0, Λ, &param_value_size);
    if (err ≠ CL_SUCCESS) {
      cerr ≪ "Failed to find OpenCL platform " ≪ str ≪ ".\n";
      return;
    }
    char *info = (char *) new char[param_value_size];
    err = clGetPlatformInfo(id, name, param_value_size, info, Λ);
    if (err ≠ CL_SUCCESS) {
      cerr ≪ "Failed to find OpenCL platform " ≪ str ≪ ".\n";
      return;
    }
    cout ≪ "\t" ≪ str ≪ ":\t" ≪ info ≪ endl;
    delete[] info;
  }
```

This code is used in section 4.

15. ⟨ Member functions of **mpoi** 6 ⟩ +≡

```
 public:
   void display_platform_info() const;
 protected:
   void _display_platform_info(cl_platform_id, cl_platform_info, string) const;
```

16. OpenCL device에 buffer 메모리를 생성할 때 지정할 속성에 관한 상수들을 정의한다.

```
⟨ Constants of mpoi 16 ⟩ ≡
public:
  enum buffer_property {
    READ_ONLY = CL_MEM_READ_ONLY,
    WRITE_ONLY = CL_MEM_WRITE_ONLY,
    READ_WRITE = CL_MEM_READ_WRITE
  };
```

This code is used in section 2.

17. OpenCL device에 메모리를 생성하고 해제하기 위한 method들을 정의한다.

```
⟨ Methods to handle device memory buffers 17 ⟩ ≡
  size_t mpoi::create_buffer(mpoi::buffer_property bp, const size_t sz) {
    cl_int err;
    cl_mem buffer = clCreateBuffer(_context, bp, sz, Λ, &err);
    _buffers[_next_key] = buffer;
    _next_key++;
    return _next_key − 1;
  }
  void mpoi::release_buffer(const size_t id) {
    if (_buffers[id] ≠ Λ) {
      clReleaseMemObject(_buffers[id]);
      _buffers[id] = Λ;
    }
  }
```

See also section 19.

This code is used in section 4.

18. ⟨ Member functions of **mpoi** 6 ⟩ +≡

```
public:
  size_t create_buffer(mpoi::buffer_property, const size_t);
  void release_buffer(const size_t);
```

19. OpenCL device의 buffer에 host의 메모리 내용을 기록하거나, 반대로 buffer의 내용을 host의 메모리로 읽어오기 위한 method.

⟨ Methods to handle device memory buffers 17 ⟩ +≡

```
void mpoi::enqueue_write_buffer(const size_t id, const size_t size, const void *mem) {
  if (_buffers[id] ≠ Λ) {
    cl_int err = clEnqueueWriteBuffer(_cmd_queue, _buffers[id], CL_TRUE, 0, size, mem, 0, Λ, Λ);
    if (err ≠ CL_SUCCESS) {
      cerr ≪ "Error in enqueuing a write buffer.\n";
      return;
    }
  }
}
void mpoi::enqueue_read_buffer(const size_t id, const size_t size, void *mem) {
  if (_buffers[id] ≠ Λ) {
    cl_int err = clEnqueueReadBuffer(_cmd_queue, _buffers[id], CL_TRUE, 0, size, mem, 0, Λ, Λ);
    if (err ≠ CL_SUCCESS) {
      cerr ≪ "Error in enqueuing a read buffer.\n";
      return;
    }
  }
}
```

20. ⟨ Member functions of **mpoi** 6 ⟩ +≡

```
public:
  void enqueue_write_buffer(const size_t, const size_t, const void *);
  void enqueue_read_buffer(const size_t, const size_t, void *);
```

21. OpenCL kernel의 인자를 설정하기 위한 method.

⟨ Methods to set kernel arguments 21 ⟩ ≡

```
void mpoi::set_kernel_argument(const size_t kernel_id, const size_t order, const size_t buffer_id) {
  if ((_kernels[kernel_id] ≠ Λ) ∧ (_buffers[buffer_id] ≠ Λ)) {
    cl_int err = clSetKernelArg(_kernels[kernel_id], static_cast⟨cl_uint⟩(order), sizeof
        (cl_mem), (void *) &(_buffers[buffer_id]));
    if (err ≠ CL_SUCCESS) {
      cerr ≪ "Error in setting a kernel argument!\n";
      return;
    }
  }
}
void mpoi::set_kernel_argument(const size_t id, const size_t order, const size_t size, void *mem)
  {
  if (_kernels[id] ≠ Λ) {
    cl_int err = clSetKernelArg(_kernels[id], static_cast⟨cl_uint⟩(order), size, mem);
    if (err ≠ CL_SUCCESS) {
      cerr ≪ "Error in setting a kernel argument!\n";
      return;
    }
  }
}
```

This code is used in section 4.

22. ⟨ Member functions of **mpoi** 6 ⟩ +≡

```
public:
  void set_kernel_argument(const size_t, const size_t, const size_t);
  void set_kernel_argument(const size_t, const size_t, const size_t, void *);
```

23. Data 병렬 처리를 위한 OpenCL kernel을 실행하는 method.

⟨ Methods to execute a kernel 23 ⟩ ≡

```
  void mpoi::enqueue_data_parallel_kernel(const size_t id, size_t num_global_items, size_t
      num_local_items) {
    if (_kernels[id] ≠ Λ) {
      for ( ; num_local_items ≠ 1; num_local_items--) {
        if (num_global_items % num_local_items ≡ 0) break;
      }
      cout ≪ "Local item size = " ≪ num_local_items ≪ endl;
      cl_int err = clEnqueueNDRangeKernel(_cmd_queue, _kernels[id], 1, Λ, &num_global_items,
          &num_local_items, 0, Λ, Λ);
      if (err ≠ CL_SUCCESS) {
        cerr ≪ "Error in enqueuing nd range kernel.\n";
        return;
      }
    }
  }
```

This code is used in section 4.

24. ⟨ Member functions of **mpoi** 6 ⟩ +≡

```
public:
  void enqueue_data_parallel_kernel(const size_t, size_t, size_t);
```

25. Test.

⟨ test.cpp 25 ⟩ ≡

```
#include "mpoi.h"
#include <iostream>
#include <chrono>
#include <cmath>
  using namespace std;
  using namespace std::chrono; int main(int argc, char *argv[]) { mpoi pc("./kernel.cl");
    pc.display_platform_info( );
    size_t kernel_id = pc.create_kernel("vec_calc");
    const size_t size = 5000000;
    float *a = (float *) new float[size];
    float *b = (float *) new float[size];
    float *c = (float *) new float[size];
    for (size_t i = 0; i ≠ size; i++) {
      a[i] = i;
      b[i] = size − i;
    }
    const unsigned num_trials = 10; vector < int > duration_parallel(num_trials); vector <
        int > duration_serial(num_trials); vector < float > difference_parallel_serial(num_trials);
    for (unsigned i = 0; i ≠ num_trials; i++) {
      auto t0 = high_resolution_clock::now( );      /* Parallel computation begins here. */
      size_t a_buffer = pc.create_buffer(mpoi::buffer_property::READ_ONLY, size * sizeof(float));
      size_t b_buffer = pc.create_buffer(mpoi::buffer_property::READ_ONLY, size * sizeof(float));
      size_t c_buffer = pc.create_buffer(mpoi::buffer_property::READ_WRITE, size * sizeof(float));
      pc.enqueue_write_buffer(a_buffer, size * sizeof(float), a);
      pc.enqueue_write_buffer(b_buffer, size * sizeof(float), b);
      pc.set_kernel_argument(kernel_id, 0, a_buffer);
      pc.set_kernel_argument(kernel_id, 1, b_buffer);
      pc.set_kernel_argument(kernel_id, 2, c_buffer);
      pc.enqueue_data_parallel_kernel(kernel_id, size, 100);
      pc.enqueue_read_buffer(c_buffer, size * sizeof(float), c);
      pc.release_buffer(a_buffer);
      pc.release_buffer(b_buffer);
      pc.release_buffer(c_buffer);
      auto t1 = high_resolution_clock::now( );      /* Parallel computation ends here. */
      duration_parallel[i] = static_cast⟨int⟩(duration_cast⟨milliseconds⟩(t1 − t0).count( ));
      float *d = (float *) new float[size];
      t0 = high_resolution_clock::now( );      /* Serial computation begins here. */
      for (size_t j = 0; j ≠ size; j++) {
        if (sin(a[j]) * cos(b[j]) > 0.F) {
          d[j] = exp(sin(a[j]) * cos(b[j]) + cos(a[j]) * sin(b[j]));
        }
        else {
          d[j] = exp(sin(a[j]) * cos(b[j]) − cos(a[j]) * sin(b[j]));
        }
      }
      t1 = high_resolution_clock::now( );      /* Serial computation ends here. */
      duration_serial[i] = static_cast⟨int⟩(duration_cast⟨milliseconds⟩(t1 − t0).count( ));
```

```
    float diff = 0.F;
    for (size_t j = 0; j ≠ size; j++) {
      diff += pow(c[j] − d[j], 2);
    }
    diff /= float(size);
    difference_parallel_serial[i] = diff;
    delete[] d;
  }
  cout ≪ "================ S U M M A R Y ================\n\n";
  float sum_ratio = 0.F;
  int sum_parallel = 0;
  int sum_serial = 0;
  cout ≪ "\tParallel" ≪ "\t\tSerial" ≪ "\t\tRatio (P/S)" ≪ "\t\tDifference\n";
  cout ≪ "  ----------------------------------------------------------------------\n";
  for (unsigned i = 0; i ≠ num_trials; i++) {
    cout ≪ "\t" ≪ duration_parallel[i] ≪ " msec" ≪ "\t\t" ≪ duration_serial[i] ≪
        " msec" ≪ "\t\t" ≪ float(duration_parallel[i])/float(duration_serial[i]) ≪ "\t\t" ≪
        difference_parallel_serial[i] ≪ endl;
    sum_parallel += duration_parallel[i];
    sum_serial += duration_serial[i];
    sum_ratio += float(duration_parallel[i])/float(duration_serial[i]);
  }
  cout ≪ "  ----------------------------------------------------------------------\n";
  cout ≪ "\t" ≪ float(sum_parallel)/float(num_trials) ≪ " msec" ≪ "\t\t" ≪
      float(sum_serial)/float(num_trials) ≪ " msec" ≪ "\t\t" ≪ sum_ratio/float(num_trials) ≪
      endl;
  delete[] a;
  delete[] b;
  delete[] c;
  return 0; }
```

26. OpenCL program.

```
⟨ kernel.cl 26 ⟩ ≡
  kernel void vec_calc(global const float *a, global const float *b, global float *result) {
    int id = get_global_id(0);
#undef __USE_CONDITIONAL__
#ifdef __USE_CONDITIONAL__
    if (sin(a[id]) * cos(b[id]) > 0.) {
      result[id] = exp(sin(a[id]) * cos(b[id]) + cos(a[id]) * sin(b[id]));
    }
    else {
      result[id] = exp(sin(a[id]) * cos(b[id]) − cos(a[id]) * sin(b[id]));
    }
#else
    result[id] = exp(sin(a[id]) * cos(b[id]) + sign(sin(a[id]) * cos(b[id])) * cos(a[id]) * sin(b[id]));
#endif
  }
```

27. Index. 이 프로그램에 사용된 심볼과 그에 대한 설명을 보려면 아래의 인덱스를 참조하라.

MPOI: Multi-Processing Object Interface

(Last revised on November 15, 2021)