

November 15, 2021 at 08:58

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

2. Declaration.

```

1  <mpoi.h 2> ≡
2  #ifndef __MULTI_PROCESSING_OBJECT_INTERFACE_H_
3  #define __MULTI_PROCESSING_OBJECT_INTERFACE_H_
4  #ifdef __APPLE__
5  #include <OpenCL/opencl.h>
6  #else
7  #include <CL/cl.h>
8  #endif
9  #include <vector>
10 #include <map>
11 #include <iostream>
12 #include <fstream>
13 #include <iterator>
14     using namespace std;
15     class mpoi {
16         < Constants of mpoi 16 >
17         < Data members of mpoi 3 >
18         < Member functions of mpoi 6 >
19     };
#endif

```

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

< Data members of mpoi 3 > ≡

```

20 protected:
21     cl_device_id _device_id;
22     cl_context _context;
23     cl_command_queue _cmd_queue;
24     cl_program _program; vector < cl_kernel > _kernels; map < size_t , cl_mem > _buffers;
25     size_t _next_key;
26     string _src;

```

This code is used in section 2.

4. Implementation.

```

27   <mpoi.cpp 4> ≡
28   #include "mpoi.h"
29   <Constructors and destructor of mpoi 12>
30   <Methods to handle mpoi object 5>
31   <Setup and cleanup of OpenCL 7>
32   <Method to build a kernel 9>
33   <Methods to display platform info 14>
34   <Methods to handle device memory buffers 17>
35   <Methods to set kernel arguments 21>
      <Methods to execute a kernel 23>

```

5. mpoi 객체를 다루기 위한 methods.

```

<Methods to handle mpoi object 5> ≡
36   mpoi &mpoi::operator=(const mpoi &obj) {
37       _next_key = 0;
38       _src = obj._src;
39       _setup_opencl();
40       if (_src != "") {
41           build_program(_src);
42       }
43       return *this;
44   }

```

This code is used in section 4.

6. <Member functions of mpoi 6> ≡

```

45   public:
46   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, &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, &Lambda);
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, &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], &Lambda);
77                 cl_uint device_vendor_id;
78                 err = clGetDeviceInfo(deviceIDs[j], CL_DEVICE_VENDOR_ID, sizeof(cl_uint),
79                                     &device_vendor_id, &Lambda);
80                 cout << "Device vendor ID: " << device_vendor_id << endl;
81                 err = clGetDeviceInfo(deviceIDs[j], CL_DEVICE_MAX_COMPUTE_UNITS, sizeof(cl_uint),
82                                     &max_compute_units, &Lambda);
83                 cout << "Device has " << max_compute_units << " compute units.\n";
84                 if (max_compute_units > current_max_compute_units) {
85                     current_max_compute_units = max_compute_units;
86                     _device_id = deviceIDs[j];
87                 }
88             }
89             delete[] deviceIDs;
90         }
91     if (!device_found) {
92         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 *)
130             &src_length, &err);
131         if (err != CL_SUCCESS) {
132             cerr << "Error in creating a program.\n";
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
153         (cl_build_status), &build_status, Λ);
154     if (build_status != CL_SUCCESS) {
155         size_t ret_val_size;
156         clGetProgramBuildInfo(_program, _device_id, CL_PROGRAM_BUILD_LOG, 0, Λ, &ret_val_size);
157         char *build_log = (char *) new char[ret_val_size + 1];
158         clGetProgramBuildInfo(_program, _device_id, CL_PROGRAM_BUILD_LOG, ret_val_size, build_log, Λ);
159         build_log[ret_val_size] = '\0';
160         cerr << "BUILD_LOG: " << build_log << endl;
161         delete[] build_log;
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),
174     _program(obj._program), _kernels(obj._kernels), _buffers(obj._buffers), _next_key(obj._next_key),
175     _src(obj._src) {}
176 mpoi::~mpoi() {
177     _cleanup_opencl();
178 }

```

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 ⟩ ≡

```

182 void mpoi::display_platform_info() const {
183     cl_uint num_platforms;
184     cl_int err = clGetPlatformIDs(0, &num_platforms);
185     if (err != CL_SUCCESS || num_platforms < 1) {
186         cerr << "Failed to find any OpenCL platforms.\n";
187         return;
188     }
189     cl_platform_id *platformIDs = (cl_platform_id *) new cl_platform_id[num_platforms];
190     err = clGetPlatformIDs(num_platforms, platformIDs, &num_platforms);
191     if (err != CL_SUCCESS) {
192         cerr << "Failed to find any OpenCL platforms.\n";
193         return;
194     }
195     cout << "Number of platforms:\t" << num_platforms << endl;
196     for (cl_uint i = 0; i != num_platforms; i++) {
197         _display_platform_info(platformIDs[i], CL_PLATFORM_PROFILE, "CL_PLATFORM_PROFILE");
198         _display_platform_info(platformIDs[i], CL_PLATFORM_VERSION, "CL_PLATFORM_VERSION");
199         _display_platform_info(platformIDs[i], CL_PLATFORM_VENDOR, "CL_PLATFORM_VENDOR");
200         _display_platform_info(platformIDs[i], CL_PLATFORM_EXTENSIONS, "CL_PLATFORM_EXTENSIONS");
201     }
202     delete[] platformIDs;
203 }
204 void mpoi::_display_platform_info(cl_platform_id id, cl_platform_info name, string str) const {
205     size_t param_value_size;
206     cl_int err = clGetPlatformInfo(id, name, 0, &param_value_size);
207     if (err != CL_SUCCESS) {
208         cerr << "Failed to find OpenCL platform" << str << ".\n";
209         return;
210     }
211     char *info = (char *) new char[param_value_size];
212     err = clGetPlatformInfo(id, name, param_value_size, info, &param_value_size);
213     if (err != CL_SUCCESS) {
214         cerr << "Failed to find OpenCL platform" << str << ".\n";
215         return;
216     }
217     cout << "\t" << str << ":\t" << info << endl;
218     delete[] info;
219 }
```

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 ⟩ ≡

```
224 public:
225     enum buffer_property {
226         READ_ONLY = CL_MEM_READ_ONLY,
227         WRITE_ONLY = CL_MEM_WRITE_ONLY,
228         READ_WRITE = CL_MEM_READ_WRITE
229     };

```

This code is used in section 2.

17. OpenCL device에 메모리를 생성하고 해제하기 위한 method들을 정의한다.

⟨ Methods to handle device memory buffers 17 ⟩ ≡

```
230     size_t mpoi::create_buffer(buffer_property bp, const size_t sz) {
231         cl_int err;
232         cl_mem buffer = clCreateBuffer(_context, bp, sz, &err);
233         _buffers[_next_key] = buffer;
234         _next_key++;
235         return _next_key - 1;
236     }
237     void mpoi::release_buffer(const size_t id) {
238         if (_buffers[id] != NULL) {
239             clReleaseMemObject(_buffers[id]);
240             _buffers[id] = NULL;
241         }
242     }

```

See also section 19.

This code is used in section 4.

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

```
243 public:
244     size_t create_buffer(buffer_property, const size_t);
245     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⟩ +≡

```

246 void mpoi::enqueue_write_buffer(const size_t id, const size_t size, const void *mem) {
247     if (_buffers[id] != NULL) {
248         cl_int err = clEnqueueWriteBuffer(_cmd_queue, _buffers[id], CL_TRUE, 0, size, mem, 0, NULL, NULL);
249         if (err != CL_SUCCESS) {
250             cerr << "Error in enqueueing a write buffer.\n";
251             return;
252         }
253     }
254 }
255 void mpoi::enqueue_read_buffer(const size_t id, const size_t size, void *mem) {
256     if (_buffers[id] != NULL) {
257         cl_int err = clEnqueueReadBuffer(_cmd_queue, _buffers[id], CL_TRUE, 0, size, mem, 0, NULL, NULL);
258         if (err != CL_SUCCESS) {
259             cerr << "Error in enqueueing a read buffer.\n";
260             return;
261         }
262     }
263 }
```

20. ⟨Member functions of mpoi 6⟩ +≡

```

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

21. OpenCL kernel의 인자를 설정하기 위한 method.

⟨Methods to set kernel arguments 21⟩ ≡

```

267 void mpoi::set_kernel_argument(const size_t kernel_id, const size_t order, const size_t buffer_id) {
268     if ((_kernels[kernel_id] != NULL) & (_buffers[buffer_id] != NULL)) {
269         cl_int err = clSetKernelArg(_kernels[kernel_id], static_cast<cl_uint>(order), sizeof
270             (cl_mem), (void *) &(_buffers[buffer_id]));
271         if (err != CL_SUCCESS) {
272             cerr << "Error in setting a kernel argument!\n";
273             return;
274         }
275     }
276     void mpoi::set_kernel_argument(const size_t id, const size_t order, const size_t size, void *mem)
277     {
278         if (_kernels[id] != NULL) {
279             cl_int err = clSetKernelArg(_kernels[id], static_cast<cl_uint>(order), size, mem);
280             if (err != CL_SUCCESS) {
281                 cerr << "Error in setting a kernel argument!\n";
282                 return;
283             }
284     }
```

This code is used in section 4.

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

```
285   void set_kernel_argument(const size_t, const size_t, const size_t);
286   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 ⟩ ≡

```
288   void mpoi::enqueue_data_parallel_kernel(const size_t id, size_t num_global_items, size_t
289     num_local_items) {
290     if (_kernels[id] != Λ) {
291       for ( ; num_local_items ≠ 1; num_local_items--) {
292         if (num_global_items % num_local_items ≡ 0) break;
293       }
294       cout ≪ "Local_item_size_=_" ≪ num_local_items ≪ endl;
295       cl_int err = clEnqueueNDRangeKernel(_cmd_queue, _kernels[id], 1, Λ, &num_global_items,
296                                             &num_local_items, 0, Λ, Λ);
297       if (err ≠ CL_SUCCESS) {
298         cerr ≪ "Error_in_enqueueing_nd_range_kernel.\n";
299         return;
300       }
300     }
```

This code is used in section 4.

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

```
301   void enqueue_data_parallel_kernel(const size_t, size_t, size_t);
```

302

25. Test.

```

⟨test.cpp 25⟩ ≡
303 #include "mpoi.h"
304 #include <iostream>
305 #include <chrono>
306 #include <cmath>
307 using namespace std;
308 using namespace std::chrono; int main(int argc, char *argv[]) { mpoi pc("./kernel.cl");
309     pc.display_platform_info();
310     size_t kernel_id = pc.create_kernel("vec_calc");
311     const size_t size = 5000000;
312     float *a = (float *) new float[size];
313     float *b = (float *) new float[size];
314     float *c = (float *) new float[size];
315     for (size_t i = 0; i ≠ size; i++) {
316         a[i] = i;
317         b[i] = size - i;
318     }
319     const unsigned num_trials = 10; vector < int > duration_parallel(num_trials); vector <
320         int > duration_serial(num_trials); vector < float > difference_parallel_serial(num_trials);
321     for (unsigned i = 0; i ≠ num_trials; i++) {
322         auto t0 = high_resolution_clock::now(); /* Parallel computation begins here. */
323         size_t a_buffer = pc.create_buffer(mpoi::buffer_property::READ_ONLY, size * sizeof(float));
324         size_t b_buffer = pc.create_buffer(mpoi::buffer_property::READ_ONLY, size * sizeof(float));
325         size_t c_buffer = pc.create_buffer(mpoi::buffer_property::READ_WRITE, size * sizeof(float));
326         pc.enqueue_write_buffer(a_buffer, size * sizeof(float), a);
327         pc.enqueue_write_buffer(b_buffer, size * sizeof(float), b);
328         pc.set_kernel_argument(kernel_id, 0, a_buffer);
329         pc.set_kernel_argument(kernel_id, 1, b_buffer);
330         pc.set_kernel_argument(kernel_id, 2, c_buffer);
331         pc.enqueue_data_parallel_kernel(kernel_id, size, 100);
332         pc.enqueue_read_buffer(c_buffer, size * sizeof(float), c);
333         pc.release_buffer(a_buffer);
334         pc.release_buffer(b_buffer);
335         pc.release_buffer(c_buffer);
336         auto t1 = high_resolution_clock::now(); /* Parallel computation ends here. */
337         duration_parallel[i] = static_cast<int>(duration_cast<milliseconds>(t1 - t0).count());
338         float *d = (float *) new float[size];
339         t0 = high_resolution_clock::now(); /* Serial computation begins here. */
340         for (size_t j = 0; j ≠ size; j++) {
341             if (sin(a[j]) * cos(b[j]) > 0.F) {
342                 d[j] = exp(sin(a[j]) * cos(b[j]) + cos(a[j]) * sin(b[j]));
343             } else {
344                 d[j] = exp(sin(a[j]) * cos(b[j]) - cos(a[j]) * sin(b[j]));
345             }
346         }
347         t1 = high_resolution_clock::now(); /* Serial computation ends here. */
348         duration_serial[i] = static_cast<int>(duration_cast<milliseconds>(t1 - t0).count());

```

```

349     float diff = 0.F;
350     for (size_t j = 0; j != size; j++) {
351         diff += pow(c[j] - d[j], 2);
352     }
353     diff /= float(size);
354     difference_parallel_serial[i] = diff;
355     delete[] d;
356 }
357 cout << "=====S_U_M_M_A_R_Y=====\\n\\n";
358 float sum_ratio = 0.F;
359 int sum_parallel = 0;
360 int sum_serial = 0;
361 cout << "\\tParallel" << "\\tSerial" << "\\tRatio_(P/S)" << "\\tDifference\\n";
362 cout << "----\\n";
363 for (unsigned i = 0; i != num_trials; i++) {
364     cout << "\\t" << duration_parallel[i] << "\\usec" << "\\t" << duration_serial[i] <<
365         "\\usec" << "\\t" << float(duration_parallel[i])/float(duration_serial[i]) << "\\t" <<
366         difference_parallel_serial[i] << endl;
367     sum_parallel += duration_parallel[i];
368     sum_serial += duration_serial[i];
369     sum_ratio += float(duration_parallel[i])/float(duration_serial[i]);
370 }
371 cout << "----\\n";
372 cout << "\\t" << float(sum_parallel)/float(num_trials) << "\\usec" << "\\t" <<
373     float(sum_serial)/float(num_trials) << "\\usec" << "\\t" << sum_ratio/float(num_trials) <<
374     endl;
375 delete[] a;
376 delete[] b;
377 delete[] c;
378 return 0;

```

26. OpenCL program.

```

<kernel.cl 26>≡
375 kernel void vec_calc(global const float *a,global const float *b,global float *result) {
376     int id = get_global_id(0);
377 #undef __USE_CONDITIONAL__
378 #ifdef __USE_CONDITIONAL__
379     if (sin(a[id]) * cos(b[id]) > 0.) {
380         result[id] = exp(sin(a[id]) * cos(b[id]) + cos(a[id]) * sin(b[id]));
381     }
382     else {
383         result[id] = exp(sin(a[id]) * cos(b[id]) - cos(a[id]) * sin(b[id]));
384     }
385 #else
386     result[id] = exp(sin(a[id]) * cos(b[id]) + sign(sin(a[id]) * cos(b[id])) * cos(a[id]) * sin(b[id]));
387 #endif
388 }

```

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

```
--APPLE__: 2.
--MULTI_PROCESSING_OBJECT_INTERFACE_H__: 2.
__USE_CONDITIONAL__: 26.
_buffers: 3, 12, 17, 19, 21.
_cleanup_opencl: 7, 8, 12.
_cmd_queue: 3, 7, 12, 19, 23.
_context: 3, 7, 9, 12, 17.
_device_id: 3, 7, 9, 11, 12.
_display_platform_info: 14, 15.
_kernels: 3, 7, 9, 12, 21, 23.
_next_key: 3, 5, 12, 17.
_program: 3, 7, 9, 11, 12.
_setup_opencl: 5, 7, 8, 12.
_src: 3, 5, 12.
_a: 25, 26.
_a_buffer: 25.
_argc: 25.
_argv: 25.
_b: 25, 26.
_b_buffer: 25.
_bp: 17.
_buffer: 17.
_buffer_id: 21.
_buffer_property: 16, 17, 18, 25.
_build_log: 11.
_build_program: 5, 9, 10, 12.
_build_status: 11.
_c: 25.
_c_buffer: 25.
_c_str: 9.
_cerr: 7, 9, 11, 14, 19, 21, 23.
_chrono: 25.
_cl_build_status: 11.
_cl_command_queue: 3.
_cl_context: 3.
_cl_device_id: 3, 7.
_CL_DEVICE_MAX_COMPUTE_UNITS: 7.
CL_DEVICE_TYPE_GPU: 7.
CL_DEVICE_VENDOR_ID: 7.
_cl_int: 7, 9, 14, 17, 19, 21, 23.
_cl_kernel: 3.
_cl_mem: 3, 17, 21.
CL_MEM_READ_ONLY: 16.
CL_MEM_READ_WRITE: 16.
CL_MEM_WRITE_ONLY: 16.
CL_PLATFORM_EXTENSIONS: 14.
cl_platform_id: 7, 14, 15.
cl_platform_info: 14, 15.
CL_PLATFORM_PROFILE: 14.
CL_PLATFORM_VENDOR: 14.
CL_PLATFORM_VERSION: 14.

cl_program: 3.
CL_PROGRAM_BUILD_LOG: 11.
CL_PROGRAM_BUILD_STATUS: 11.
CL_SUCCESS: 7, 9, 11, 14, 19, 21, 23.
CL_TRUE: 19.
cl_uint: 7, 14, 21.
clBuildProgram: 9.
clCreateBuffer: 17.
clCreateCommandQueue: 7.
clCreateContext: 7.
clCreateKernel: 9.
clCreateProgramWithSource: 9.
clEnqueueNDRangeKernel: 23.
clEnqueueReadBuffer: 19.
clEnqueueWriteBuffer: 19.
clFinish: 7.
clFlush: 7.
clGetDeviceIDs: 7.
clGetDeviceInfo: 7.
clGetPlatformIDs: 7, 14.
clGetPlatformInfo: 14.
clGetProgramBuildInfo: 11.
clReleaseCommandQueue: 7.
clReleaseContext: 7.
clReleaseKernel: 7.
clReleaseMemObject: 17.
clReleaseProgram: 7.
clSetKernelArg: 21.
cos: 25, 26.
count: 25.
cout: 7, 14, 23, 25.
create_buffer: 17, 18, 25.
create_kernel: 9, 10, 25.
current_max_compute_units: 7.
d: 25.
device_found: 7.
device_vendor_id: 7.
deviceIDs: 7.
diff: 25.
difference_parallel_serial: 25.
display_platform_info: 14, 15, 25.
duration_cast: 25.
duration_parallel: 25.
duration_serial: 25.
endl: 7, 11, 14, 23, 25.
enqueue_data_parallel_kernel: 23, 24, 25.
enqueue_read_buffer: 19, 20, 25.
enqueue_write_buffer: 19, 20, 25.
err: 7, 9, 11, 14, 17, 19, 21, 23.
exit: 7, 9.
exp: 25, 26.
```

false: 7.
get_global_id: 26.
global: 26.
high_resolution_clock: 25.
i: 7, 14, 25.
id: 9, 14, 17, 19, 21, 23, 26.
ifstream: 9.
in: 9.
info: 14.
is_open: 9.
istreambuf_iterator: 9.
j: 7, 25.
kernel: 26.
kernel_id: 21, 25.
length: 9.
main: 25.
map: 3.
max_compute_units: 7.
mem: 19, 21.
milliseconds: 25.
mpoi: 2, 5, 6, 7, 9, 12, 13, 14, 17, 18, 19, 21, 23, 25.
name: 9, 14.
now: 25.
num_devices: 7.
num_global_items: 23.
num_local_items: 23.
num_platforms: 7, 14.
num_trials: 25.
obj: 5, 12.
order: 21.
param_value_size: 14.
pc: 25.
platformIDs: 7, 14.
pow: 25.
push_back: 9.
READ_ONLY: 16, 25.
READ_WRITE: 16, 25.
release_buffer: 17, 18, 25.
result: 26.
ret_val_size: 11.
set_kernel_argument: 21, 22, 25.
sign: 26.
sin: 25, 26.
size: 7, 9, 19, 21, 25.
src: 9, 12.
src_file: 9.
src_length: 9.
src_string: 9.
std: 2, 25.
str: 14.
string: 3, 9, 10, 12, 13, 14, 15.
sum_parallel: 25.

⟨ Check kernel program build error 11 ⟩ Used in section 9.
⟨ Constants of **mpoi** 16 ⟩ Used in section 2.
⟨ Constructors and destructor of **mpoi** 12 ⟩ Used in section 4.
⟨ Data members of **mpoi** 3 ⟩ Used in section 2.
⟨ Member functions of **mpoi** 6, 8, 10, 13, 15, 18, 20, 22, 24 ⟩ Used in section 2.
⟨ Method to build a kernel 9 ⟩ Used in section 4.
⟨ Methods to display platform info 14 ⟩ Used in section 4.
⟨ Methods to execute a kernel 23 ⟩ Used in section 4.
⟨ Methods to handle device memory buffers 17, 19 ⟩ Used in section 4.
⟨ Methods to handle **mpoi** object 5 ⟩ Used in section 4.
⟨ Methods to set kernel arguments 21 ⟩ Used in section 4.
⟨ Setup and cleanup of OpenCL 7 ⟩ Used in section 4.
⟨ **kernel.cl** 26 ⟩
⟨ **mpoi.cpp** 4 ⟩
⟨ **mpoi.h** 2 ⟩
⟨ **test.cpp** 25 ⟩

MPOI: Multi-Processing Object Interface

(Last revised on November 15, 2021)

	Section	Page
Introduction	1	1
Declaration	2	2
Implementation	4	3
Test	25	12
Index	27	14

Copyright © 2016–2021 by Changmook Chun

This document is published by Changmook Chun. All rights reserved. No part of this publication may be reproduced, stored in a retrieval systems, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author.