OpenCL introduction

OpenCL

- Data and task parallel model
- Derived from the ISO C99 standard
  - With parallel extensions
- Numeric operations performed according to the IEEE754 standard
- Support of embedded and mobile devices
- Data transfer between OpenGL, OpenGL ES

See OpenCL specification for details

OpenCL

- Heterogeneous platform support
  - Parallel CPU cores
  - GPU
  - Digital Signal Processor (DSP)
  - Cell/B.E. processor

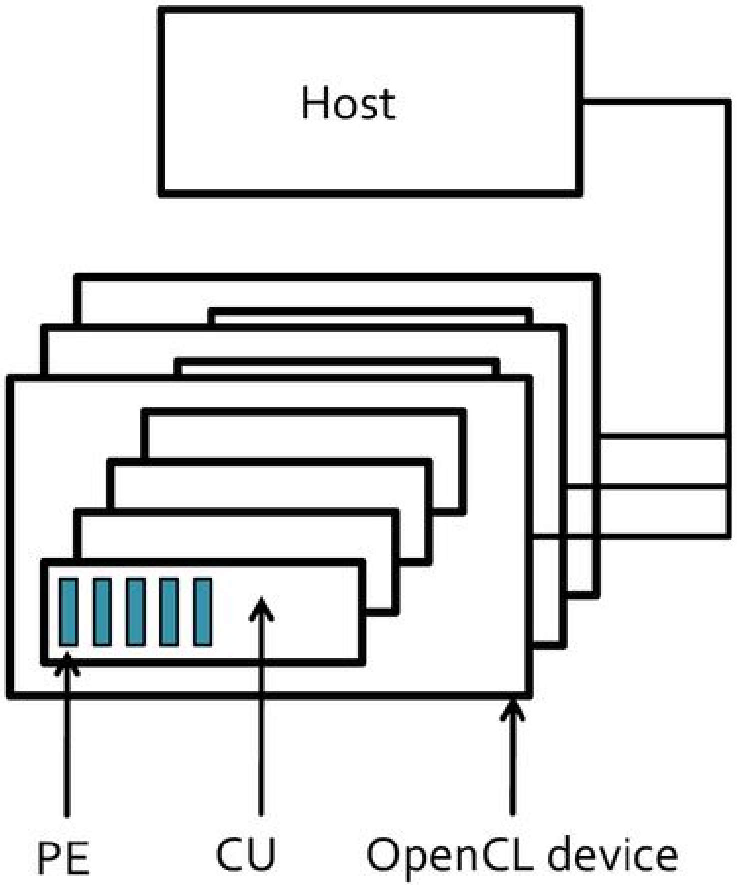
OpenCL Architecture

- Elements of the OpenCL architecture
  - Platform model
  - Execution model
  - Memory model
  - Programming model

Platform model

- Host device
- OpenCL device

- Computing Unit (CU)
  - Processing Element(PE)
    - Single Instruction Multiple Data
      SIMD (common program counter)
    - Single Program Multiple Data
      SPMD
      (independent program counters)



Execution model

- Host program
  - Context management
  - Execution control
- Kernel program
  - Controlling the CUs

Execution model

- Kernel program
  - Index space (NDRange)
  - Work-groups
  - Work-items
    - global ID
    - Same programs in the work-group
    - The execution control can differ in different units

Execution model

- Kernel program
  - Index space (NDRange)
  - Work-groups
    - Finer indexing mechanism
    - Work-group ID
    - Local ID for the Work-Items
  - Work-Items

Execution model

- Kernel program
  - Index space (NDRange)
    - N dimensional problem space (N=1,2,3)
    - Each index has the same dimensionality
  - Indexing
    - Global index space: (G_x, G_y)
    - Size of work-groups: (S_x, S_y)
      - Work-group ID (w_x, w_y)
    - Local ID (s_x, s_y)

Execution model

- Calculating IDs
  - Global address space: (G_x, G_y)
  - Work-group size: (S_x, S_y)
    - Work-group ID: (w_x, w_y)
  - Local ID: (s_x, s_y)

  - Global ID: $(g_x, g_y) = (w_x \cdot S_x + s_x, w_y \cdot S_y + s_y)$
  - Number of work-groups: $(W_x, W_y) = (G_x / S_x, G_y / S_y)$
  - Work-group ID: $(w_x, w_y) = ((g_x - s_x) / S_x, (g_y - s_y) / S_y)$

Execution model

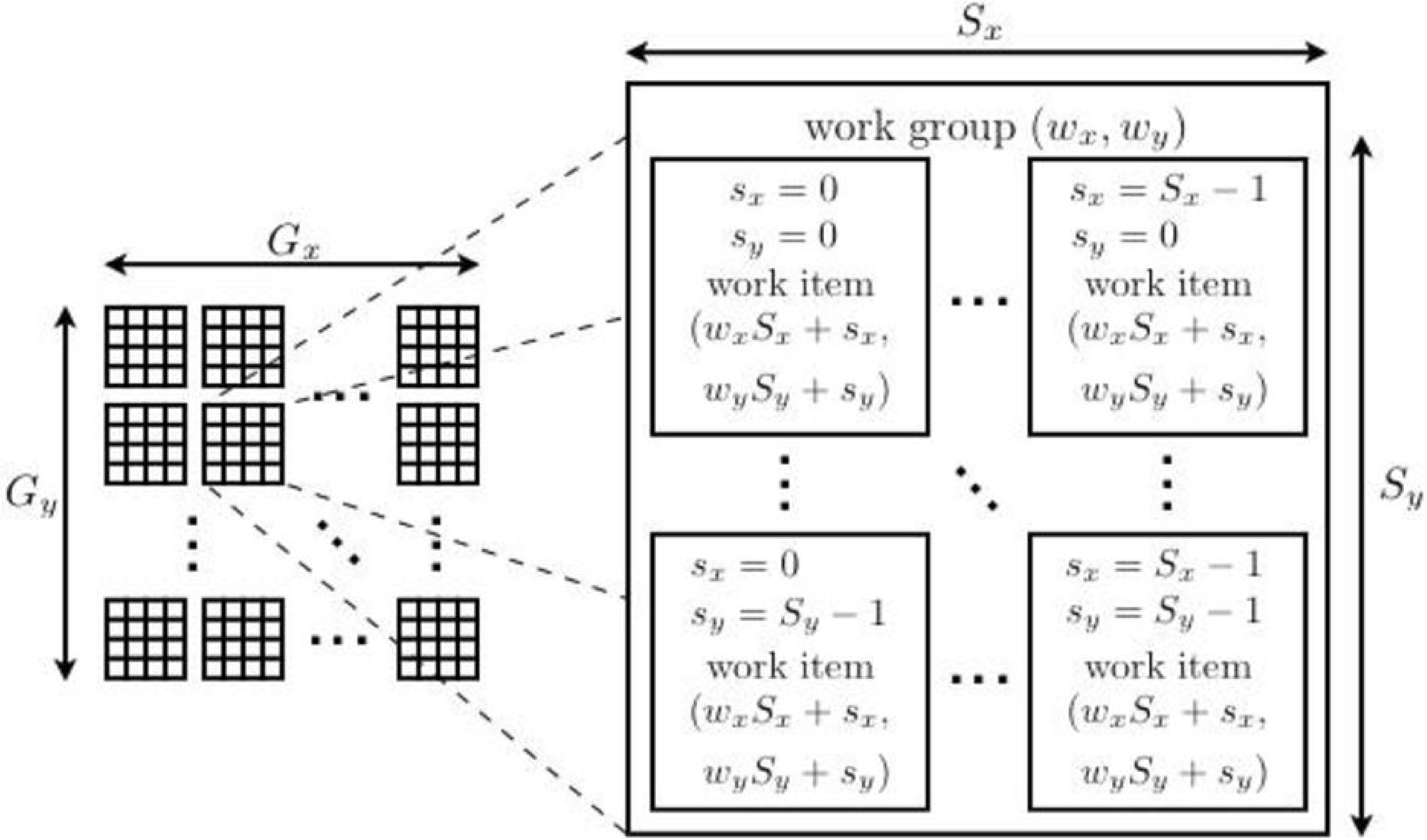


Image: Johannes Rudolph's blog

Execution model

- Context
  - Devices: a set of OpenCL capable devices
  - Kernels: a set of OpenCL functions
  - Program objects:
    - Kernel source code
    - Executable binary representation
  - Memory objects:
    - Memory used by the host and the OpenCL devices
    - Other values seen by the kernels

Execution model

- Command-queue
  - A command stream controlled by the host
  - Controls the execution of the threads
  - Commands:
    - Kernel execution
    - Memory operations
    - Synchronization

Execution model

- Command-queue execution modes
  - In-order execution
    - FIFO
    - Serializes the execution order of commands in a queue
    - a prior command on the queue completes before the following command begins
  - Out-of-order execution
    - Commands are issued in order, but do not wait to complete before following commands execute
    - Any order constraints are enforced by the programmer through explicit synchronization commands.

Execution model

- Kernel types
  - OpenCL kernel
    - OpenCL C functions
    - Executable on an OpenCL device
  - Native kernel
    - Functions accessed through host function pointer
    - Can share memory objects with OpenCL kernels
    - Optional support

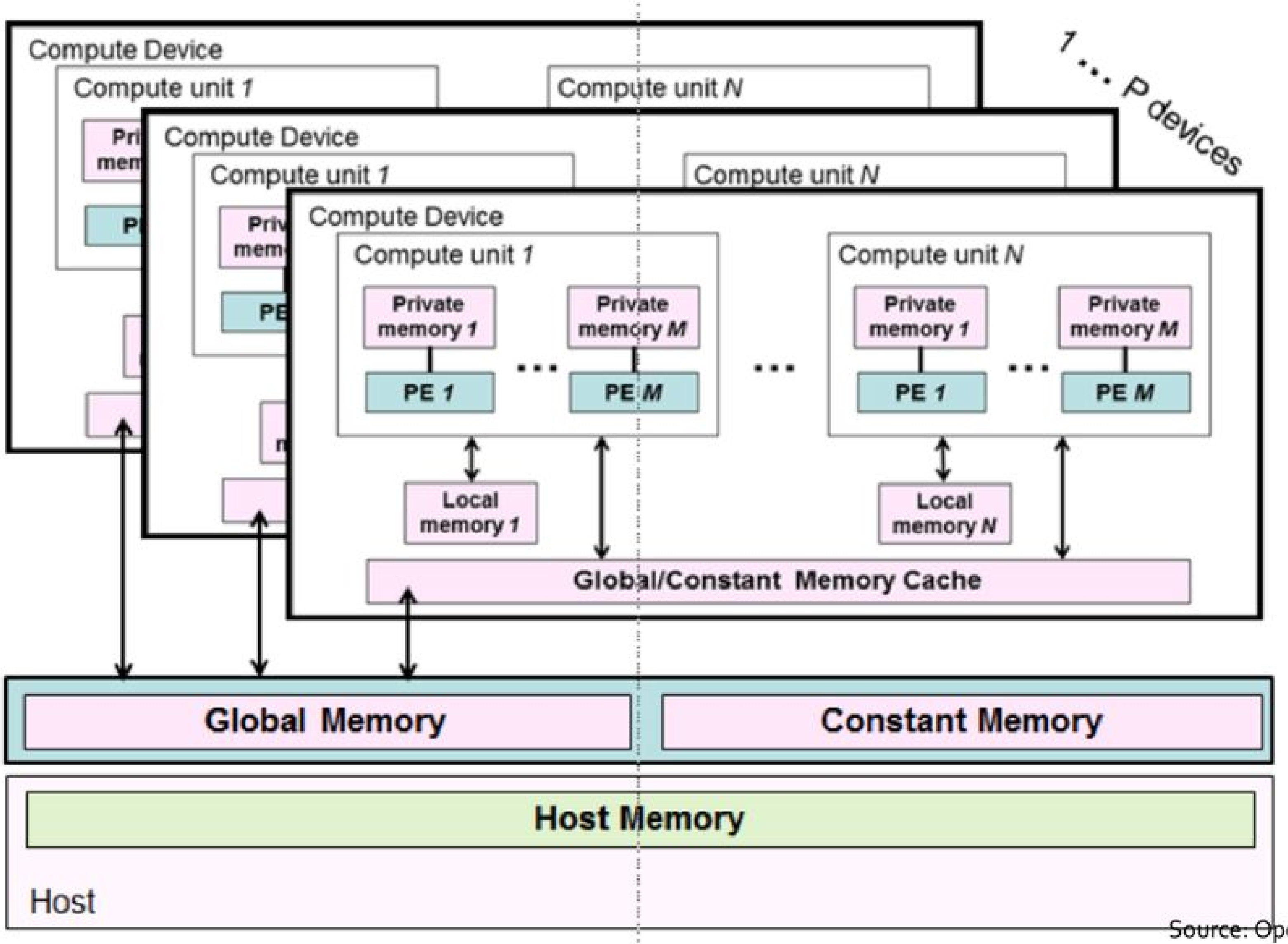
Memory model

- Four distinct memory regions are available
  - Global memory
    - Work-items can read or write any element of it
    - Allocated by the host
  - Constant memory
    - Remains constant during the execution of the kernel
    - The host allocates and initializes it
    - Can be defined statically in the kernel

Memory model

- Four distinct memory regions are available
  - Local memory
    - Shared memory within a work-group
    - Each work-item in a work-group can read or write it
    - Not visible from the host
    - May be implemented as dedicated regions of memory
  - Private memory
    - A region of memory private to a work-item
    - Seen only by the work-item

Memory model



Source: OpenCL Specification

Memory model

- The global memory is handled by the host
  - Memory allocation
  - Copy data to the memory objects
    - Synchronous and asynchronous operations
  - Release of memory objects

  - The global memory can be mapped into the host memory address space

Memory model

- Relaxed Consistency
  - The state of memory, visible to a work-item, is not guaranteed to be consistent across the collection of work-items at all times
  - Within a work-item memory has load / store consistency
  - Local memory
    - Consistent in a single work-group
  - Global memory
    - Consistent in a single work-group
    - No guarantees of memory consistency between different work-groups
  - Consistency for memory objects shared between enqueued commands is enforced at a synchronization point

Programming model

- Data parallel model
  - Defines a computation in terms of a sequence of instructions applied to multiple elements of a memory object
  - The index space defines the work-items and how the data maps on the work-items
  - Not restricted to one-to-one mapping

  - Hierarchical data parallelism
    - Explicit model
      - Total number of work-items and their division into work-groups
    - Implicit model
      - Only the number of work-items is specified the division into work-groups is automatic

Programming model

- Task parallel model
  - The kernel has only one instance
  - Independent from the index space

  - Operations on vector types
  - Multiple independent tasks
  - Enqueuing native kernels to run them in parallel

Synchronization

- Synchronization within a work-group
  - Synchronize work-items
  - work-group barrier
    - Blocking call
    - Each work-item has to reach the barrier before any are allowed to continue
  - There is no mechanism for synchronization between work-groups

Synchronization

- Synchronization in a command-queue
  - In case of out-of-order execution
  - command-queue barrier
    - Ensures that all previously queued commands have finished execution
    - Resulting updates to memory objects are visible to subsequently enqueued commands
    - Cannot be used to synchronize between command-queues
  - Waiting on an event
    - Each function generates an event that identifies the command and memory objects it updates
    - The execution of command can be suspended until the occurrence of some events

OpenCL C

- Scalar types
  - bool
  - unsigned char, char (8 bit integer)
  - unsigned short, short (16 bit integer)
  - unsigned int, int (32 bit integer)
  - unsigned long, long (64 bit integer)
  - float (IEEE754 floating-point)
  - half (16 bit float)
  - size_t (return type of the sizeof operator 32/64 bit)
  - ptrdiff_t (difference between two pointers 32/64 bit)
  - (u)intptr_t (pointer type)
  - void

OpenCL C

- Vector types
  - (u)char*n*
  - (u)short*n*
  - (u)int*n*
  - (u)long*n*
  - float*n*

  - The signed values represented in two's complement form
  - (u) stands for unsigned
  - n can be 2,4,8,16

OpenCL C

- Vector components
  - Swizzle operator (.xyzw)
    - float4 f; f.xy; f.xxyy;
  - Numeric indices (.s[0-9|a-f|A-F])
    - float4 f; f.s12;
    - float16; f.saBcdE
  - Halving (.odd, .even, .lo, .hi)
    - float4 f; f.hi; f.even.lo;
    - float4 left, right;
      float8 interleaved;
      interleaved.even = left; interleaved.odd = right;

OpenCL C

- Conversion between different types
  - Implicit conversion
    - Limited usability
    - Between scalar types
  - Explicit conversion
    - Scalar – Vector conversion
      - float4 f = (float4)1.0;
    - Conversion between vector types
      - destType*n* convert_destType_sat_roundingMode(sourceType*n*)
        - _sat – truncation to the codomain
        - _roundingMode – rounding
      - uchar4 u; int4 c = convert_int4(u);

OpenCL C

- Conversion between types
  - Types should have same size
  - as_type*n*()
    - float f = 1.0f;
      uint u = as_uint(f); // the value will be: 0x3f800000
    - float4 f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);
      int4 i = as_int4(f);
      // (0x3f800000, 0x40000000, 0x40400000, 0x40800000)

OpenCL C

- Address space qualifiers
  - __global : global memory
    - __global float4 color;
  - __local : local memory
    - __local float16 shared;
  - __contant : constant memory
    - __constant float uniformData;
    - Can be initialized from the host
  - __private : private memory
    - __private float8 workItemExclusive;

OpenCL C

- Function qualifiers
  - __kernel : OpenCL function
    - Only an OpenCL device can execute it
    - The host program can call it
    - Other OpenCL kernels can call it
  - __attribute__ : hints to the compiler
    - vec_type_hint(type*n*) : size of vector operations
      - Work-items can be merged or separated by the compiler to better match the hardware capabilities

OpenCL C

- Built-in functions
  - Work-item information:
    - uint get_work_dim()
    - size_t get_global_size(uint dimIdx);
    - size_t get_global_id(uint dimIdx);
    - size_t get_local_size(uint dimIdx);
    - size_t get_local_id(uint dimIdx);
    - size_t get_num_groups(uint dimIdx);
    - size_t get_group_id(uint dimIdx);

OpenCL C

- Built-in functions
  - Math functions
    - E.g. sin, cos, tan, floor …
    - float, half, integer types
  - Common functions
    - E.g. clamp, min, max …
    - float types
  - Geometric functions
    - E.g. cross, dot, length, normalize …
    - float types
  - Relational functions
    - E.g. isequal(float*n*, float*n*)
      isfinite(float)
    - float types

OpenCL C

- Built-in functions
  - Vector load functions
    - pointer – vector conversion
  - Vector store functions
    - Vector – pointer conversion

OpenCL C

- Built-in functions
  - Synchronization functions
    - barrier(flag);
      - All work-items in a work-group must execute this function before any are allowed to continue
      - CLK_LOCAL_MEM_FENCE : makes the local memory consistent
      - CLK_GLOBAL_MEM_FENCE : makes the global memory consistent
    - mem_fence(flag);
      - Loads and stores will be committed to memory
    - read_mem_fence(flag);
    - write_mem_fence(flag);

OpenCL C

- Built-in functions
  - Async Copy functions
    - From global memory to local memory
    - From local memory to global memory
    - event_t async_work_group_copy(...);
    - wait_group_events(..., eventList);
  - Prefetch
    - Loads a part of the global memory to the cache
  - Atomic functions
    - E.g. atomic_add

My first OpenCL program

```
#include <iostream>
#include <CL/opencl.h>

#define DATA_SIZE (1024*1240)

int main(int argc, char* argv[]){
  cl_int err;

  size_t global;  // global space
  size_t local;   // local space

  cl_platform_id platform;
  err = clGetPlatformIDs(1, &platform, NULL);
  if(err != CL_SUCCESS){
    std::cerr << "Error: Failed to find a platform!" << std::endl;
    return EXIT_FAILURE;
  }

//...
```

My first OpenCL program

```
  cl_device_id device_id;
  err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device_id, NULL);
  if(err != CL_SUCCESS){
    std::cerr << "Error: Failed to create a device group!" << std::endl;
    return EXIT_FAILURE;
  }

  cl_context context;
  context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);
  if (!context) {
    std::cerr << "Error: Failed to create a compute context!" << std::endl;
    return EXIT_FAILURE;
  }

  cl_command_queue commands;
  commands = clCreateCommandQueue(context, device_id, 0, &err);
  if (!commands) {
    std::cerr << "Error: Failed to create a command commands!" << std::endl;
    return EXIT_FAILURE;
  }
// ...
```

My first OpenCL program

```
  cl_program program;
  program = clCreateProgramWithSource(context, 1,
                            (const char **) &KernelSource, NULL, &err);
  if (!program) {
    std::cerr << "Error: Failed to create compute program!" << std::endl;
    return EXIT_FAILURE;
  }

  err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
  if (err != CL_SUCCESS) {
    size_t len;
    char buffer[2048];

    std::cerr << "Error: Failed to build program executable!" << std::endl;
    clGetProgramBuildInfo(program, device_id, CL_PROGRAM_BUILD_LOG,
                          sizeof(buffer), buffer, &len);
    std::cerr << buffer << std::endl;
    exit(1);
  }

// ...
```

My first OpenCL program

```
  cl_kernel kernel;
  kernel = clCreateKernel(program, "square", &err);
  if (!kernel || err != CL_SUCCESS) {
    std::cerr << "Error: Failed to create compute kernel!" << std::endl;
    exit(1);
  }

  float* data = new float[DATA_SIZE];    // input array
  float* results = new float[DATA_SIZE]; // output array
  unsigned int correct;
  cl_mem input;                        // device memory object for the input
  cl_mem output;                       // device memory object for the output

  // the input values are random
  unsigned int count = DATA_SIZE;
  for(int i = 0; i < count; i++){
    data[i] = rand() / (float)RAND_MAX;
  }

// ...
```

My first OpenCL program

```
  input = clCreateBuffer(context,  CL_MEM_READ_ONLY, sizeof(float) * count,
                                                                     NULL, NULL);
  output = clCreateBuffer(context, CL_MEM_WRITE_ONLY,sizeof(float) * count,
                                                                     NULL, NULL);
  if (!input || !output) {
    std::cerr << "Error: Failed to allocate device memory!" << std::endl;
    exit(1);
  }

  // copy input values to the global memory of the device
  err = clEnqueueWriteBuffer(commands, input,
                             CL_TRUE, 0, sizeof(float) * count,
                             data, 0, NULL, NULL);
  if (err != CL_SUCCESS) {
    std::cerr << "Error: Failed to write to source array!" << std::endl;
    exit(1);
  }

// ...
```

My first OpenCL program

```
  // Kernel arguments
  err = 0;
  err  = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
  err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
  err |= clSetKernelArg(kernel, 2, sizeof(unsigned int), &count);
  if (err != CL_SUCCESS) {
    std::cerr << "Error: Failed to set kernel arguments! " << err << std::endl;
    exit(1);
  }

  // Setting up the work-group size
  err = clGetKernelWorkGroupInfo(kernel, device_id,
                                  CL_KERNEL_WORK_GROUP_SIZE,
                                  sizeof(local), &local, NULL);
  if (err != CL_SUCCESS) {
    std::cerr << "Error: Failed to retrieve kernel work group info! "
              <<  err << std::endl;
    exit(1);
  }
// ...
```

My first OpenCL program

```
  // Enqueuing the kernel
  global = count;
  err = clEnqueueNDRangeKernel(commands, kernel,
                               1, NULL, &global, &local,
                               0, NULL, NULL);
  if (err) {
    std::cerr << "Error: Failed to execute kernel!" << std::endl;
    return EXIT_FAILURE;
  }

  // Waiting for the kernel to be executed
  clFinish(commands);
  // Reading the result
  err = clEnqueueReadBuffer( commands, output,
                             CL_TRUE, 0, sizeof(float) * count,
                             results, 0, NULL, NULL );
  if (err != CL_SUCCESS) {
    std::cerr << "Error: Failed to read output array! " <<  err << std::endl;
    exit(1);
  }
// ...
```

My first OpenCL program

```
  // Validating the result
  correct = 0;
  for(int i = 0; i < count; i++) {
    if(results[i] == data[i] * data[i])
      correct++;
  }

  std::cout << "Computed " << correct << "/" <<
               count << " correct values" << std::endl;
  std::cout << "Computed " << 100.f * (float)correct/(float)count <<
               "% correct values" << std::endl;

// ...
```

My first OpenCL program

```
    // Cleaning
    delete [] data; delete [] results;

    clReleaseMemObject(input);
    clReleaseMemObject(output);
    clReleaseProgram(program);
    clReleaseKernel(kernel);
    clReleaseCommandQueue(commands);
    clReleaseContext(context);

    return 0;

}
```

My first OpenCL program

- OpenCL kernel

```
__kernel void square(
                     __global float* input,
                     __global float* output,
                    const unsigned int count){
  int i = get_global_id(0);
  if(i < count){
    output[i] = input[i] * input[i];
  }
}
```